AFRL-IF-RS-TR-2002-234
Final Technical Report
September 2002

HIGH PERFORMANCE REAL-TIME FUSION ARCHITECTURE

Integrated Sensors, Inc.

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2002-234 has been reviewed and is approved for publication.

APPROVED:

STEVEN L. DRAGER
Project Engineer

FOR THE DIRECTOR:

MICHAEL L. TALBERT, Major, USAF
Technical Advisor
Information Technology Division
Information Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 2002	**Final** **Jul 00 – Sep 01**

4. TITLE AND SUBTITLE

HIGH PERFORMANCE REAL-TIME FUSION ARCHITECTURE

5. FUNDING NUMBERS

C - F30602-00-C-0111
PE - 63789F
PR - 407T
TA - HR
WU - PT

6. AUTHOR(S)

Garry Fountain

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Integrated Sensors, Inc.
502 Court Street, Suite 210
Utica, NY 13502

8. PERFORMING ORGANIZATION REPORT NUMBER

N/A

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

AFRL/IFTC
26 Electronic Pky
Rome NY 13441-4514

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

AFRL-IF-RS-TR-2002-234

11. SUPPLEMENTARY NOTES

AFRL Project Engineer: Steven Drager/IFTC/(315) 330-2735/Steven.Drager@rl.af.mil

12a. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 Words)***

The objective of this effort was to define, prototype and demonstrate an affordable next-generation JDL Level 2 fusion and exploitation architecture called the High Performance Real-Time Fusion Architecture (HPRTF). It was the goal of HPRTF to produce the following:

-- Identification of processing bottlenecks that limit current Level 2 fusion systems from performing in real-time
-- Development of a hardware requirements trade space for real-time execution and identification of affordable solutions.
-- Definition of an affordable, scalable, next generation fusion architecture, which will support real-time execution of Level 2 fusion systems.
-- Development and demonstration of critical portions of a next generation Level 2 fusion architecture prototype system that demonstrates the capability of real-time execution.

14. SUBJECT TERMS	15. NUMBER OF PAGES
JDL Level 2 Fusion, fusion architecture, system evaluation, parallelization, simulation testing	69
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Table of contents

List of Figures

1. Introduction

A constant challenge in automating the field of state-of-the-art data fusion systems to aid and amplify the operator's ability is to process data in or near real-time and produce prompt results to the operator. The speed of any fusion process is critical in defining its usefulness in time-sensitive missions. The current-day battlefield surveillance operator is faced with a data reduction task where enemy movements amongst civilian traffic must be manually identified and correlated with other off-board surveillance systems to better understand unfolding events as they are happening.

Modern fusion systems often suffer poor speed performance as their advanced correlation techniques and np-hard computations are more compute intensive than single-processor computers can handle. These systems generate valuable products but require a great amount of time to complete. Therefore the battlefield situation is known hours later or in some cases the next day. Consider the impact of a real-time solution allowing a complex fusion system to function in or near real-time. Multiple-processor, or parallel computers offer a tremendous performance potential but come at a price. The cost of purchasing and maintenance of such a machine can be expensive. Further, implementing and/or porting the fusion algorithms onto these machines can add significant costs.

The purpose of this effort was to define, prototype, and demonstrate an affordable next-generation JDL Level 2 fusion and exploitation architecture called the High Performance Real Time Fusion Architecture (HPRTF). The scope of this effort includes the identification of the limiting criteria in current Level 2 fusion assessment systems that inhibit them from providing real-time assessment. Then, using this criterion and an assessment of the hardware requirements trade space, determine and propose the required solutions. The trade-space of hardware and software solutions will include system cost and maintenance among other factors. The proposed solutions will consist of both hardware and software approaches including an assessment of cost impact. The software developed will be made portable across multiple hardware architectures. The appropriateness of the solutions will then be validated by applying them to an existing state-of-the-art Level 2 fusion system, the Nodal Exploitation Application Toolkit (NEAT) developed by Sterling Software; Rome, NY. The effort will culminate in a demonstration of the prototype HPRTF system using multiple data scenarios.

The effort succeeded in providing a demonstration of the prototype HPRTF system using multiple data scenarios on a Linux Cluster at the Air Force Research Laboratory's Rome Research Site, in Rome, NY. Performance improvements were made by parallelizing existing algorithms across multiple processors. However the sequential algorithmic processing was optimized as well. Compared to the original software running on a Linux PC, results show an average of 47 times speed improvement on a single processor and a 95 times speed improvement when run in parallel across 4 CPUs. A development environment was also installed to facilitate integration and reuse of the HPRTF products with other projects after completion of the HPRTF contract.

1.1 Introduction to Fusion

The Department of Defense Joint Directors of Laboratories (JDL) have defined data fusion as a multilevel, multifaceted process dealing with the automatic detection, association, correlation, estimation, and combination of data and information from single and multiple sources. This multilevel model classifies data fusion into four categories.

Level 1 Fusion accomplishes object refinement, or the process of fusing detection-level data to entity tracks. These tracks contain state-vector information concerning the location and velocity of a single target and optionally some form of identification, based on the sensor type.

Level 2 Fusion refers to situation refinement and assessment, or the process of fusing Level-1 tracks into groups based on physical location and target identities. Situation assessment is accomplished by identifying the general purpose or Military Unit Id of the target groups based on a rule-based or template-based association.

Level 3 Fusion is threat refinement and assessment, which is the process of fusing the Level-2 situation assessment products and related capability data to infer intent of these target groups.

Level 4 Fusion refers to process refinement, which monitors the performance of the first 3 Levels and redirects surveillance assets to improve performance of the overall fusion process and thereby increasing the quality of the data.

1.2 Nodal Exploitation and Analysis Tools – An Introduction to NEAT

The function of NEAT is an automation of what is currently done by operators correlating and associating intelligence reports by hand. This is currently done by either secure radio between surveillance platforms or ground-stations, or days or weeks after the data was collected. The amount and quality of correlation and identification of military units by this method is based solely on the resourcefulness and military knowledge of the operators examining the intelligence reports. There is a clear need for an automation of this process to allow operators to both cover greater areas and produce quicker processing times.

NEAT is a JDL Level-2 Fusion system, which means that it has the ability to locate, classify, and track military units. It accomplishes this by passing data and control parameters through a sequence of processing events. This process is outlined in Figure 1.

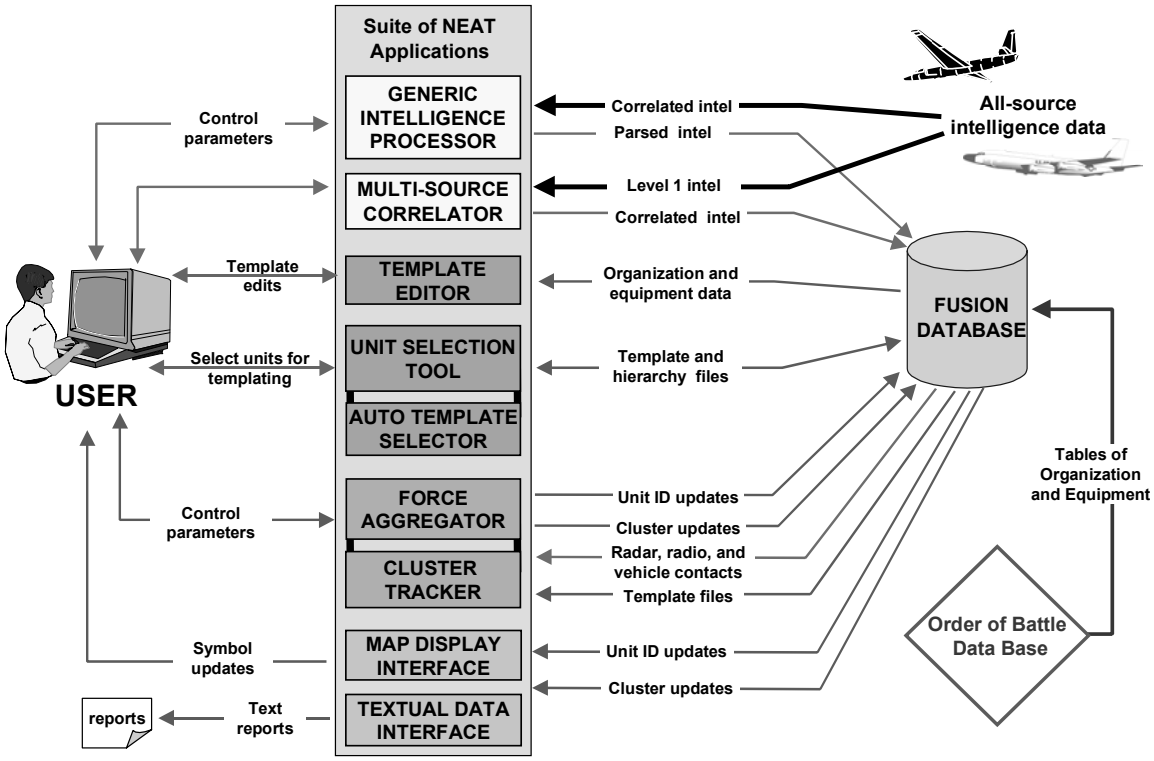


Figure 1- NEAT Application Suite (Sterling Software)

First, a user supplies a list or set of templates for NEAT to use. Templates contain equipment and organization descriptions and layout that define military unit types. The selected template set is effectively the search criteria for all further processing. The operator uses the **Unit Selection Tool** to choose military units of interest and form these templates. The selected template set can either be a single-depth list of military unit types or a "tree" of unit types referred to as a template hierarchy. Optionally, the user can use the **Template Editor** to form new templates or modify existing ones.

With these selected templates in place, NEAT can begin to process intelligence reports. Level-1 reports from offboard sources are first parsed and formatted by the **Generic Intelligence Processor**. The properly formatted reports are then inserted into the Fusion Database used by NEAT. The **Multi-Source Correlator** then reads these Level-1 reports and divides them into correlated lists of radar, radio, and vehicle contact reports.

At this point a "scene" of data is ready to be processed by the **Force Aggregator**. A scene refers to a snapshot of activity over a battlefield for a specific time interval. This time interval varies with the surveillance assets used, the geographic size of the area covered, and amount of activity within the area.

Once a scene of data is collected, the Force Aggregator compares the contact reports with templates either chosen by the user or selected automatically from a hierarchical tree of templates. Based on the associations of the contact reports with the template set and other contact reports, clusters of reports are merged to form nodes. These nodes are classified based on the content of their group, using the best-fit military template. A track is established for the node by the **Cluster Tracker** and is updated as additional scenes of data are received.

Once processing is completed, the final node reports are displayed on the **Map Display Interface**. The display allows detailed examination of the output of the Force Aggregator including what contact reports are contained in each node.

NEAT fulfills the role of a Level-2 Fusion system but cannot execute in real-time, and could take minutes to several hours to process a single scene of Level-1 intelligence reports. Therefore the usage of NEAT is currently limited to off-line use (non-real-time) to process the reports after the fact.

NEAT would become a powerful tool if it could process Level-1 reports in real-time. Operating from a surveillance aircraft or ground-station, NEAT could monitor military communication networks for real-time Level-1 intelligence feeds (see Figure 2). It would quickly locate, classify, and track military units and publish the C2 Nodes across these networks to the operators that need them. This would provide a situation-awareness on the battlefield that has never been realized before. Furthermore, by use of custom templates entered by the operator, it could be used as a trip-wire system to monitor military and non-military activity for a specific event over an entire area of interest.

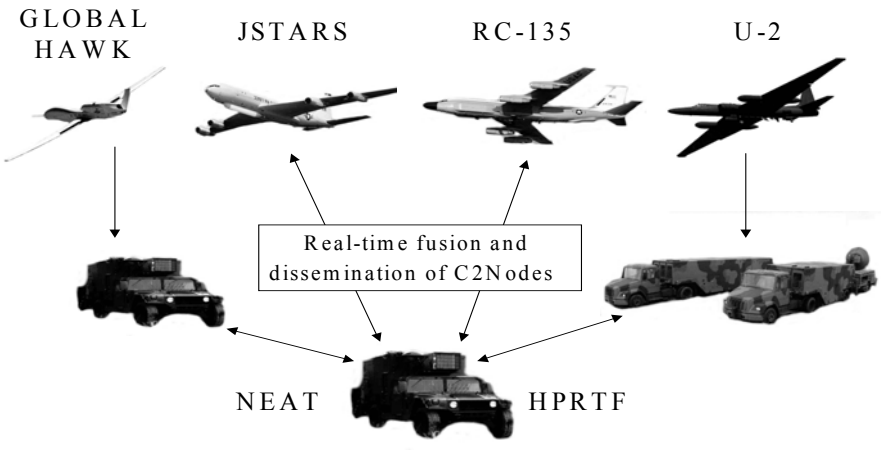


Figure 2- Potential of "Real-Time" NEAT

1.3 The Force Aggregator

The unique feature of NEAT is the Force Aggregator, which is the centerpiece application in the suite of NEAT tools. It includes a number of innovative algorithms to support Level 2 fusion of correlated all-source intelligence data. These include capabilities to fuse Level 1 radar, radio, and vehicle contact reports, and then using templating techniques based on Bayesian probability theory, classify and identify military units and C2 nodes. The Force Aggregator includes a situation display capability providing the operator access to the results of NEAT Level 2 assessment. The map-based graphical display shows the probable location of NEAT-identified military units, as well as all contacts that contribute to the identity.

Figure 3 below depicts the process flow diagram of the NEAT Force Aggregator component.

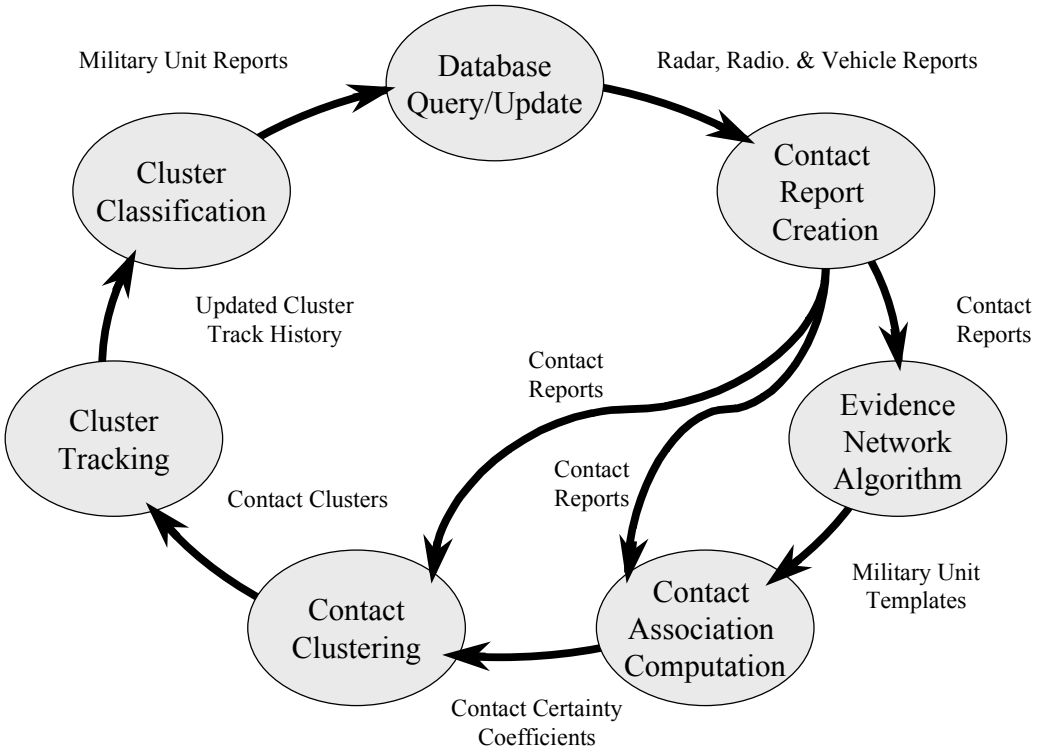


Figure 3 – Force Aggregation Process Flow (Sterling Software)

A short description of each of the NEAT Force Aggregator components follows:

- **Database Query** - This is the first step in the force aggregation process that obtains a list of radar, radio, and vehicle intelligence reports produced by Level 1 Fusion applications. These intelligence reports can be filtered by geographic region, timeliness, allegiance, affiliation, and so on. A Sybase database is used by NEAT to store and access data.

- **Contact Report Creation** – The next step in the Force Aggregation process flow creates a table of contact reports from the intelligence reports. The contact reports are a reduction of the Intel reports to an efficient format for subsequent classifier processing.

- **Evidence Network Algorithm** – This is an algorithm for military unit template set selection. A military unit template is a data structure that describes the composition of a military unit and contains arguments used by a Bayesian classifier such as prior probabilities and observation search radii. The goal of the Evidence Network Algorithm is to select, given the available intelligence data, a military unit template set that is strongly correlated with the current contact report set and that is consistent with the probability space partitioning rules of Bayesian probability.

- **Contact Association Computation** – The fourth step in the force aggregation process is to take the contact reports created from step 2, and the template set selected from step 3, and use the Bayesian classifier to compute contact association coefficients. These coefficients express the relative strengths of the pair-wise relationships between contacts. The association coefficients are later normalized and biased to express a certainty that the contacts of a contact pair are or are not members of the same contact cluster.

- **Current Clusters** – at this point, contacts are grouped into clusters of mutually related contacts given the certainty coefficients computed in step 4.This is accomplished using a clustering algorithm which is based upon a greedy clustering algorithm.

- **Cluster Tracking** – The sixth step in the force aggregation process flow is to perform Cluster Tracking. The cluster tracker is based on a "Bayesian sonar post-detection acoustic contact data fusion algorithm" that Sterling Software adapted to the ground cluster-tracking problem.

- **Cluster Classification** - The final algorithmic step of the force aggregation process is to perform cluster classification, where a military unit classification is assigned to each cluster given the cluster posterior probabilities and military unit prior probabilities.

- **Database Update** - The final overall step in the force aggregation process is the Database Update procedure, where new node records are inserted into the Sybase database and the radar, radio, and vehicle intelligence reports are updated to reflect new entity associations as identified by the Level 2 fusion process.

1.4 Scenario definitions

Simulated data was used to feed Level 1 Fusion products to NEAT. Many data sets or scenarios were used during this effort for investigating performance bottlenecks, both in the existing algorithms and testing/validating the optimized parallel revisions in downstream tasks. Of these, three main scenarios were chosen to exercise different algorithmic steps within the Force Aggregator. A brief explanation of these scenarios follow.

- **618th SAM (Surface to Air Missile) Regiment Scenario (SAM-618)**
  This scenario contains 161 total intelligence reports, consisting of 51 radar reports, 110 vehicle reports, and no radio reports. A template file is used to populate 12 military unit templates for all force aggregation processing. In this usage of NEAT, the operator is searching for a SAM regiment within a geographic area, and only SAM templates are considered. This is the smallest scenario and requires just minutes to execute (single CPU, serial version), which makes it a good candidate for software validation as it provides minimal testing time.

- **TRAC R6 Division Scenario (TRAC-R6)**
  This intermediate-sized scenario contains 1,465 intelligence reports spread across 162 radar, 3 radio, and 1300 vehicle reports. A template file containing 18 military unit templates is used. This scenario contains a motorized rifle division of considerable size and distribution, and specifically impacts the current clustering algorithm due to the large number of spontaneous clusters the scenario generates.

- **TRAC R61 Division Scenario (SCEN-R61)**
  This scenario is the largest data set considered in this effort, with 1531 intelligence reports in 168 radar, 4 radio, and 1359 vehicle reports. This is basically a motorized rifle regiment with the addition of SAM fortifications. Templates are passed in the form of a Hierarchy file containing a total of 185 military unit templates. This is a usage of NEAT where the operator wants an intelligence preparation of the battlefield, and does not necessarily know what ground units exist. This heavily stresses the contact association coefficients processing step by greatly increasing the number of templates considered.

2. High Performance Real-Time Fusion (HPRTF)

This section will introduce the analysis done to NEAT and work performed to build the prototype HPRTF architecture. The process-intensive algorithms from NEAT will be identified, studied, and ported into the HPRTF architecture to allow possible real-time Level-2 Fusion. It was the goal of HPRTF to produce the following:

- Identification of processing bottlenecks that limit current Level 2 fusion systems from performing in real-time.
- Development of a hardware requirements trade space for real-time execution.
- Identification of affordable solutions.
- Detailed definition of an affordable, scalable, next generation fusion architecture, which will support real-time execution of Level 2 fusion systems.
- Development and demonstration of critical portions of a next generation Level 2 Fusion architecture prototype system that demonstrates the capability of real-time execution.

To analyze the code to determine where most processing time was spent, the Integrated Sensors Inc. "RTExpress" product was used as a development, testing, and visualization environment. Additionally, data transport and distribution libraries in RTExpress were augmented and used to define the HPRTF system. The top-level Force Aggregation function was written in Matlab™, with all sub-components of the force aggregation process divided into Matlab-accessible C functions. This allowed easy source compilation and visualization of performance across multiple processing nodes using RTExpress and complex data visualization using Matlab graphs and plots. Further, rapid prototyping of various data distribution and processing concepts were accomplished at the Matlab level without modifying the algorithm source code.

2.1 Processing Bottlenecks and Solutions

Of the main processes contained in Figure 1, The Force Aggregator component is the main limiting factor restricting the NEAT system from providing real-time situation refinement. This component contains the entire Level-2 Fusion engine used by NEAT. Thus under this task the primary function within NEAT that will be analyzed is the Force Aggregator component. The remaining components are not time constraints, rather they serve to configure the Force Aggregator with templates, feed Level-1 contact reports into it, and display the resulting nodes as output. In addition, it was not in the scope of this effort to re-write the display portion of NEAT so the graphic interface was completely reused.

Later sub-sections will address how each component in the Force Aggregator works and what processing bottlenecks exist in NEAT. Each bottleneck identified will include a data-flow diagram to aid in the understanding of its function. While this section will detail force aggregation processing bottlenecks within NEAT, it should be noted that NEAT serves only as a test case to the HPRTF architecture. HPRTF and its assumptions and requirements should be generic to all Level 2 Fusion systems.

2.1.1 The Sybase Bottleneck

NEAT uses a Sybase database as a back plane to manage contact report data and the resulting fused products. Sybase is not a real-time database and represents a significant non-processing bottleneck. If reduced, significant time savings could be gained. Access to Sybase on some HPC platforms may be difficult and slow, and there is no desire to tie a Sybase dependency into HPRTF. Sterling Software had originally implemented a data cache system to regain some of the performance lost to Sybase. This cache system could be reused to form a shell around the Force Aggregation code to both allow the algorithms to function with minimal code changes and allow execution without the use of Sybase (see Figure 4). Eventually HPRTF will be interfaced to the original NEAT system. This will require a file or socket connection to allow two-way communication between NEAT and HPRTF hosts. These requirements will be kept in mind while evaluating hardware architecture solutions in section 3.

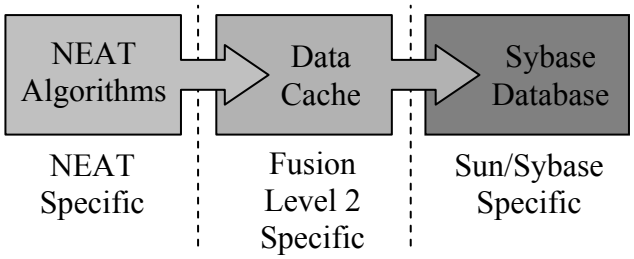


Figure 4 – NEAT Data Access Layers

2.1.2 Sybase solutions - Data Transport Mechanism HPC-Link

To satisfy both a demonstration layout where both NEAT and HPRTF systems communicate in real-time such as with sockets over Ethernet LAN, and an engineering/testing setup where the same inputs can be run through HPRTF while rapidly changing fusion parameters and code development; the HPC-link module was created.

HPC-link exists as a C library that allows the transfer of Level-2 Fusion I/O between software applications running on the same or different machines, with varying operating systems and endian-order representations.

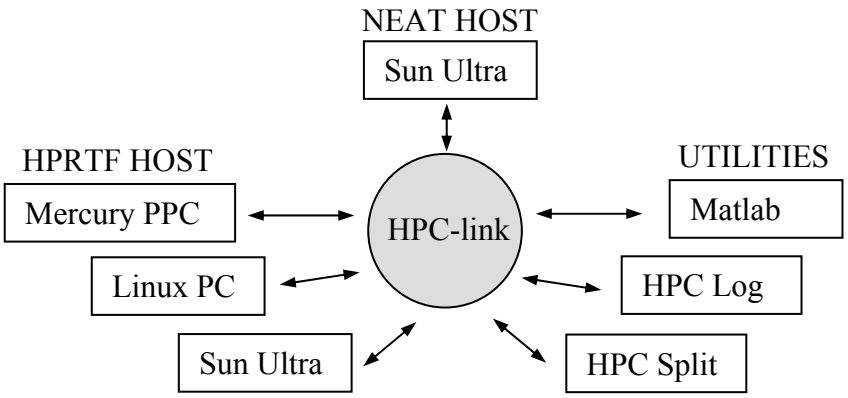


Figure 5 – HPC-link and Utilities

Additionally, a small set of visualization tools was developed to construct and verify test case cache files on both the input and output side of the force aggregator. A "log" program was written which flattens the binary cache file into understandable ASCII text. The NEAT debugging printout format was used to clarify the data contents during discussions with Sterling Software. The log application also served as a line-by-line data porting check for implementation on hardware architectures with various data alignment differences. A "split" function was also developed to divide NEAT-generated contact report cache files into sub-cache files containing segregated data. These sub-cache files could then be reassembled using simple Unix shell utilities by concatenating the binary files into a new test case. For example, the split application would separate a particular cache file into radar, radio, vehicle, and node files. By reassembling the files without the node reports from the previous run, a static "no history" scene could be generated. Or perhaps the individual files could be investigated using the log application or plotting data values in Matlab. Figure 5 summarizes data flow in and out of HPC-link.

2.1.3 Data Distribution and Collection Library

The first version of the stand-alone Force Aggregator was completed shortly after the completion of HPC-link and related utilities. This was a single-processor configuration which simply wrappered most of the Force Aggregation process under one Matlab function wrapper. A data transport software package was now required to facilitate execution across multiple processing nodes. This library augments the core RTExpress message-passing layer which utilizes MPI for all low-level operations. These augmentations were required due to the complexity and size of data associated with the Level 2 Fusion process. Figure 6 is a table of these data management functions.

DispListInfo	Displays an entire list of Level 1 Intelligence records in ASCII for a given cache mxArray.
DispRecord	Displays one Level 1 Intelligence record in ASCII, given the cache mxArray and report index.
DistScatter	Scatters data by columns using RTExpress™ redistribute functions. **Case 1: Group-leader-only** - data is dispensed from the group leader to every member in the group. A full copy of the data matrix is sent to all. **Case 2: Local mxArray -** each CPU has a total copy of the mxArray and uses redistribute to slice the mxArray by columns. There are only local transfers in this case. Data is not transmitted between processing nodes.
LeadColAdd	Will perform a column add on the group-leader-only matrix. Matrices from worker nodes are converted to sparse matrices before collected by the group leader.
LeadGather	Gathers data that has been distributed (in columns) to the group leader. Will also concatenate data together that is not distributed to the group leader.
LocalGather	Will gather data that is distributed or local to every group member. Will also concatenate data together that is not distributed to every group member.
LocalScatter	Scatters data from the Group Leader to every member in the group where the data is distributed by columns from the input mxArray to the output mxArray. This function only scatters data that is not already distributed.
MatrixLoad	Loads any binary mxArray form from disk file.
MatrixStore	Stores any binary mxArray data to disk file.
Sparse	Creates a sparse matrix from a full matrix. Binary data can be passed as the full matrix and data compression can be applied.

Figure 6 – Data Distribution and Collection Functions

2.1.4 Evidence Network Algorithm

The Evidence Network Algorithm is a composite, nested C++ object consisting of 8,000+ total lines of source code. It is the first step in a chain of processing events that forms the Force Aggregation function. Its main purpose simply stated is to select a reduced set of templates given a list of contact reports as input. Observation vectors are formed at each contact report location, class-conditional probability is computed for each template, and a collectively exhaustive and mutually exclusive template list is produced given the input contact set. This process is illustrated in figure 7. This was designed to alleviate the processing load involved in all downstream template-based computations by reducing the amount of templates in the processing cache.

When first investigated, processing time spent in the Evidence Network Algorithm was exaggerated by equipment and organization look-up function calls that used costly array-search loops. This originally accounted for up to 77% of the total time for a run. This was immediately optimized by studying how the table look-ups were used, limiting their usage, and optimizing the functions themselves. After which, it was found that 2-32% of processing time was spent in the Evidence Network. This wide time range was a result of the two processing modes in the Evidence Network, template set and template hierarchical processing. "Template set" refers to a single list of templates specified by the operator for force aggregation. In this mode, the Evidence Network Algorithm is largely bypassed and the fusion system is used to find and identify military unit types of interest to the operator. In the Hierarchical mode, an entire hierarchy of templates is passed to the fusion system and the Evidence Network is utilized to its full potential by selecting templates within the template tree that relate to contact reports in the scenario. The remaining templates from the hierarchical list are discarded.

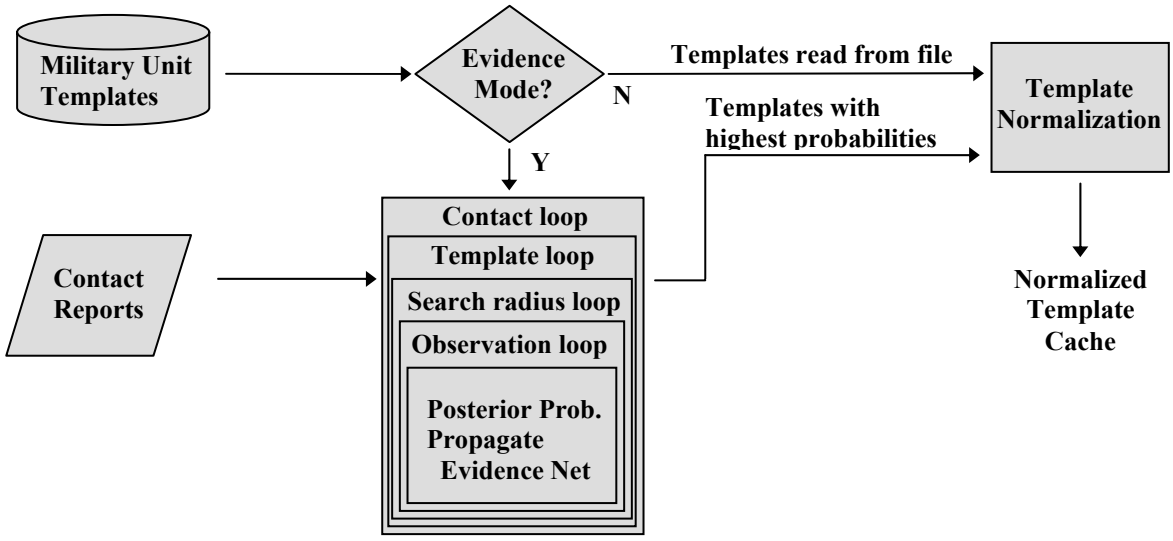


Figure 7 – Evidence Network – First Look

2.1.5 Evidence Network Solutions

The portion of the Evidence Network that best offered itself to parallelization is the observation step where observation vectors are generated for every contact report. The observation-processing loop also constitutes nearly 90% of processing time in the Evidence Network Algorithm when in template-hierarchical mode. While in this mode, a data-parallel method was chosen for implementing this parallel step where the contact report list is initially divided across all processing nodes for the observation vector computation. In the case of a template list mode, where the operator has already pre-selected the templates, this processing step is skipped all together. In either case the complete template set is required on every processing node and is distributed by file using the original NEAT access functions. The results are then merged on the group leader (processing node 0) and the remaining portion of the algorithm runs sequentially. Figure 8 below illustrates the data flow of the parallel Evidence Network Algorithm.

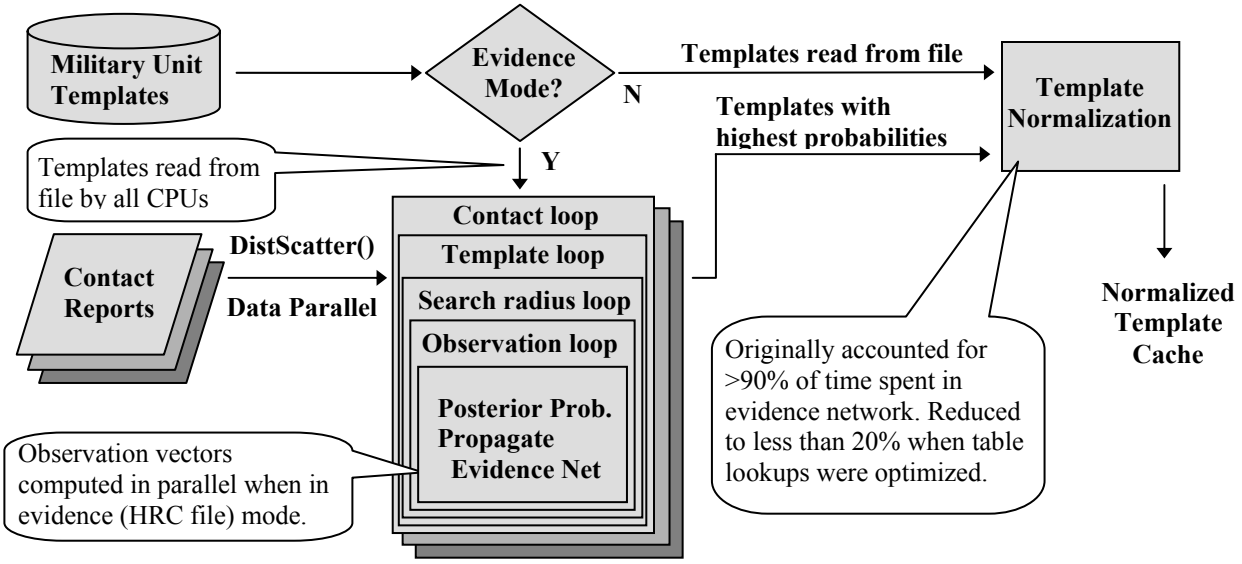


Figure 8 – The Parallel Evidence Network Algorithm

The Evidence Network Algorithm also utilizes table look-up functions for military unit organization and equipment references. When possible, these function calls were optimized and in some cases the usage of the organization and equipment tables were aligned such that only one table search occurred per template.

The greatest amount of table look-ups occur in the final "Template Normalization" step. This originally accounted for more than 90% of the time spent in the evidence network, reduced to 20% after optimizations. Figure 9 illustrates these processing time improvements realized by sequential and parallel optimizations.

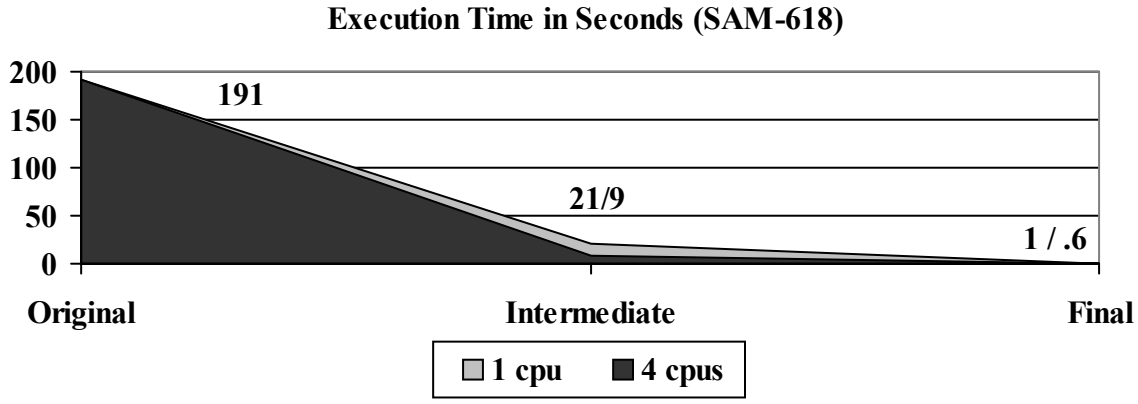


Figure 9 – Evidence Network Performance Improvements

2.1.6 Contact Association Coefficients

After optimizing the evidence network algorithm and the associated table look-ups, the "Create Contact Association Coefficients" step was responsible for the bulk of time (65-72%) spent in processing. This was a 600-line brute-force nested loop where observation vectors were formed for every contact report, class-conditional probability computed for each template, and the posterior probability of each military unit combined to form a contact association matrix. Figure 10 shows the nesting of each processing loop.

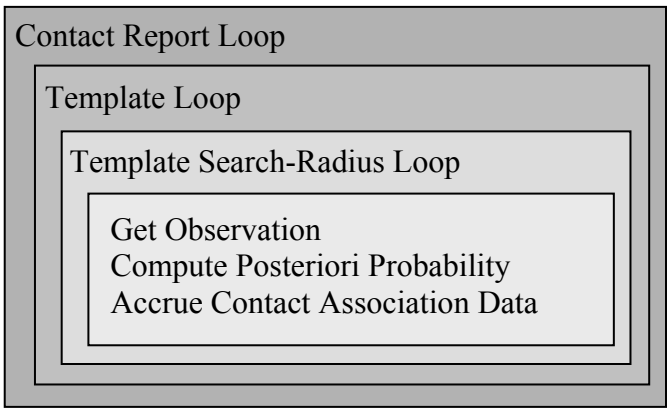


Figure 10 – Contact Associations – First Look

This computation was necessary as every contact report ultimately needed to be compared with every template. Processing time could be reduced by optimizing the inner-most loops where most execution time was spent, or perhaps swapping the contract loop with the template loop if necessary, but this component best offers itself to parallelization. A data-parallel method here would seem to be optimal.

2.1.7 Contact Association Coefficients Solutions

The solution used to implement this step was to distribute the contact report list across all processing nodes in a data-parallel manor and streamline the observation and association-data access functions. Alternative approaches were also tested, distributing the templates in data-parallel and processing the entire contact list across all CPUs, but this did not align well with the neighboring fore and aft algorithms which required all templates passed to all processing nodes.

When this algorithm was first investigated, it was found to contain sequential array/table searches that exaggerated its slow execution speed. These sequential look-up functions were changed to hash-tables and the surrounding data access functions re-written to use references instead of passing data. The use of data references on the innermost loop resulted in a 50% speed-up of that section alone. Additionally, usage and walking of linked-lists were avoided when possible to reduce execution time further. The "GetObservation" function contained many linked-list access calls. A 30% speed improvement was achieved by avoiding linked-list access when possible. Figure 11 shows where in the processing steps these optimizations occurred and Figure 12 summarizes performance improvements. Processing time for this function was reduced by more than 50x (583 to 11 seconds in the Trac_R6 scenario) once all functions were optimized. The majority of the time improvement resulted from the hash table implementation.

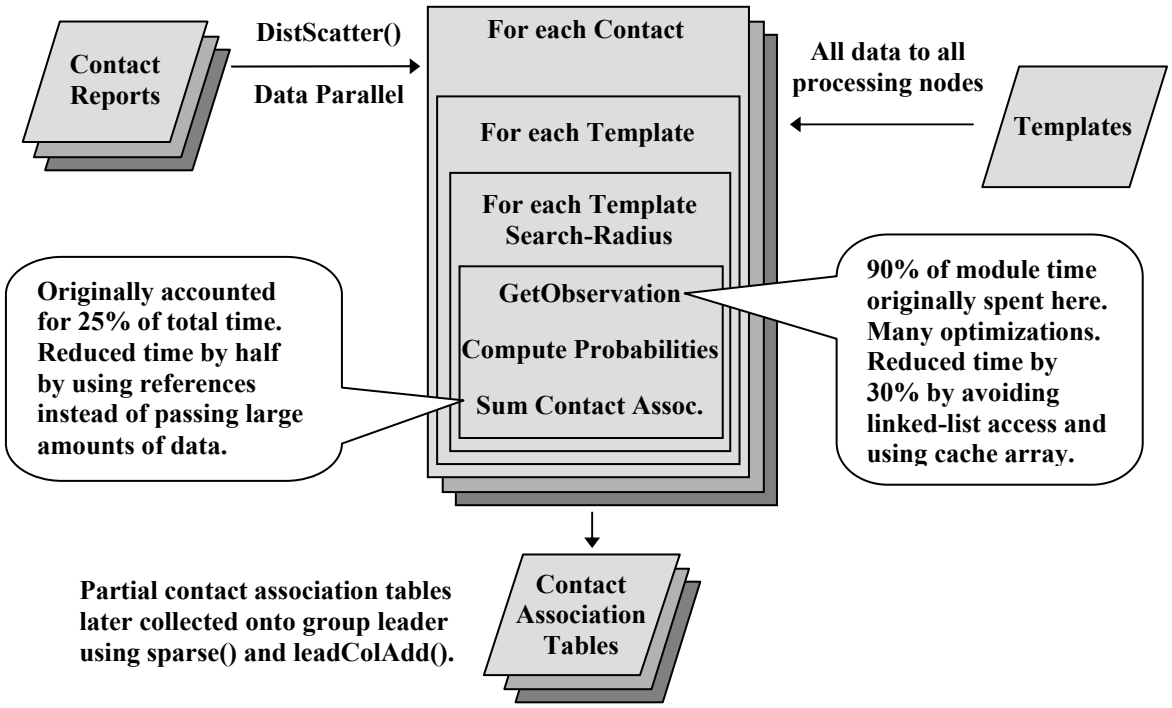


Figure 11 – Parallel Contact Association Coefficients Computation

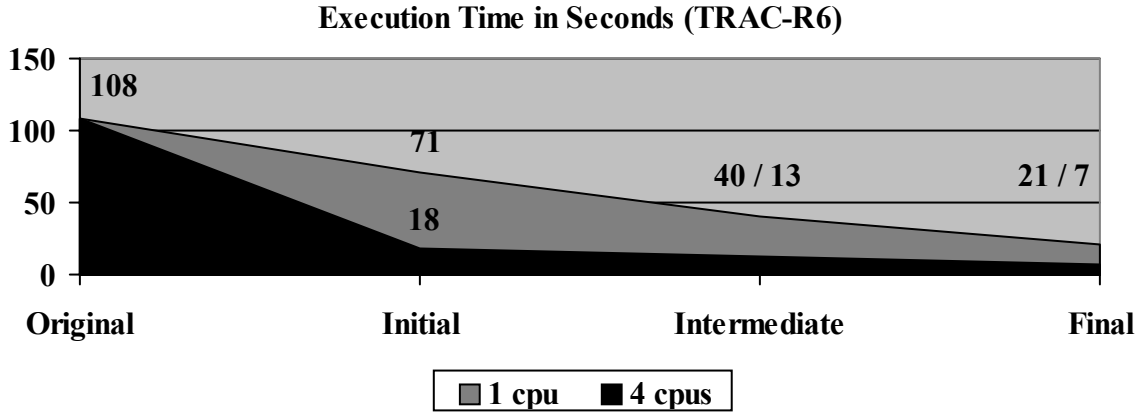


Figure 12 – Contact Association Performance Improvements

2.1.8 Build Current Clusters

The "Current Clustering" Algorithm represented the most difficult step to parallelize. This 2000 plus line module was responsible for building clusters from the current contact report set using the contact certainty coefficients (normalized from the contact association coefficients). It was the largest memory consumer of all force aggregation sub-components and could grow between 5% to 80% of execution time depending on scenario content. Low memory conditions and disk/memory swapping were responsible for longer execution durations. The Current Clustering Algorithm contained two main processing steps. First, a "coarse clustering" processing loop occurs where the contact certainty table was traversed and clusters were spawned for all contact reports. A significant amount of memory (in some cases 200+ Mb) was used at this point as there existed as many clusters as contact reports and each cluster contained multiple contact report lists. This processing loop used the majority of time spent in current clustering, and needed to have access to all clusters it generated as the loop iterated, which did not immediately lend itself easily to parallelization. The second loop merged these spontaneous clusters and formed the current cluster table. These two main processing steps are show in Figure 13.

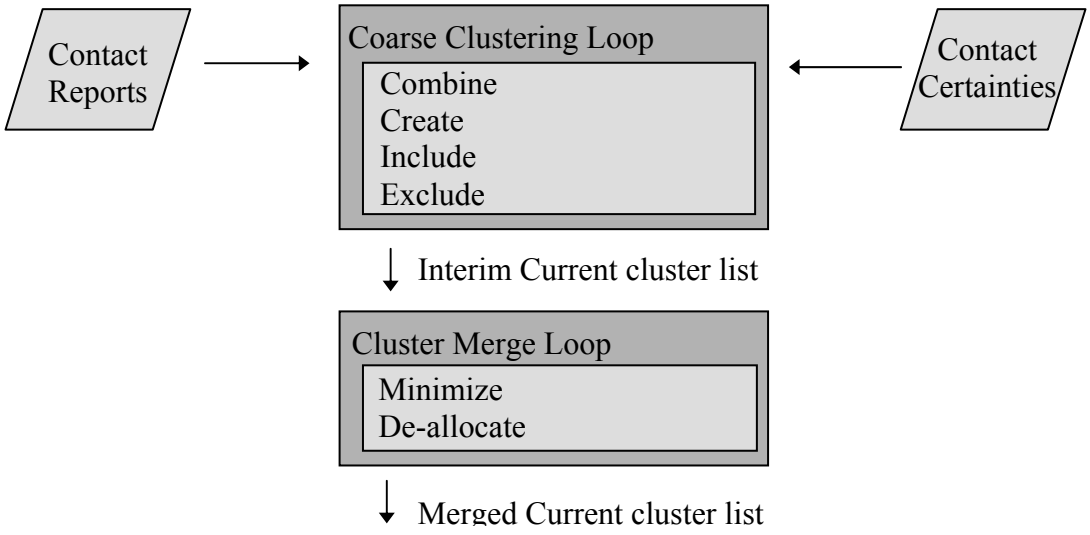


Figure 13 – Current Clusters – First Look

2.1.9 Build Current Clusters Solutions

This component is identified as a high-bandwidth step and shouldn't lend itself easily to parallelization. However an alternative solution was tested using a parallel coarse clustering function but no improvement was gained. Due to the nature of the algorithm and amount of memory it requires, it was decided that this step should remain sequential on a single CPU. With optimization in mind, Sterling Software re-wrote this module to function several times faster than the original. In addition, hash-table functions developed for the Contact Association Coefficient Computation were reused in this step to greatly reduce the "coarse clustering" step. The optimizations occurred on inner processing loops and resulted in big improvements, reducing time spent in "Coarse Clustering" from 10 minutes to 11 seconds. Memory usage was still a factor, and the algorithm was later modified to use a much reduced-size contact-report list format that resulted in this step using a tenth of the memory than it used before. This was possible because many fields in the contact report structure were unused or utilized larger-than-necessary data types. Figure 14 documents sequential optimizations made to this Force Aggregator component. Final performance improvement was more than 10x over the original optimized version (1.4 seconds using the Trac_R6 scenario, see Figure 15).

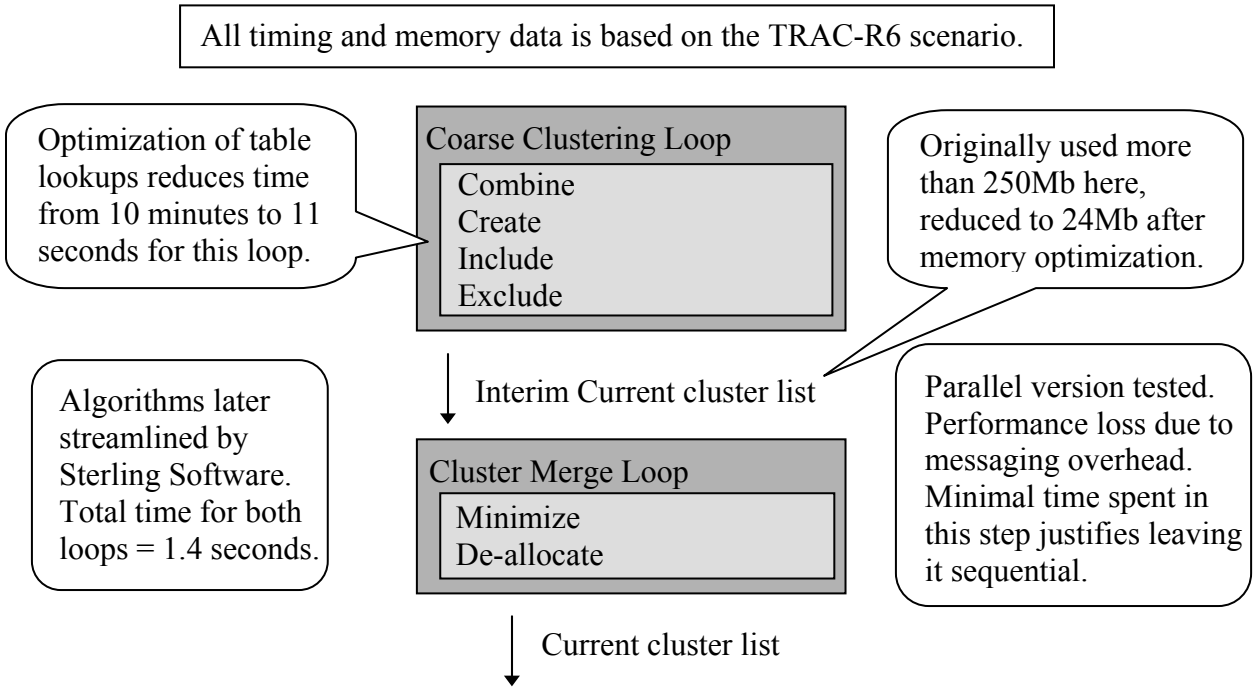


Figure 14 – Optimized Current Clustering Algorithm

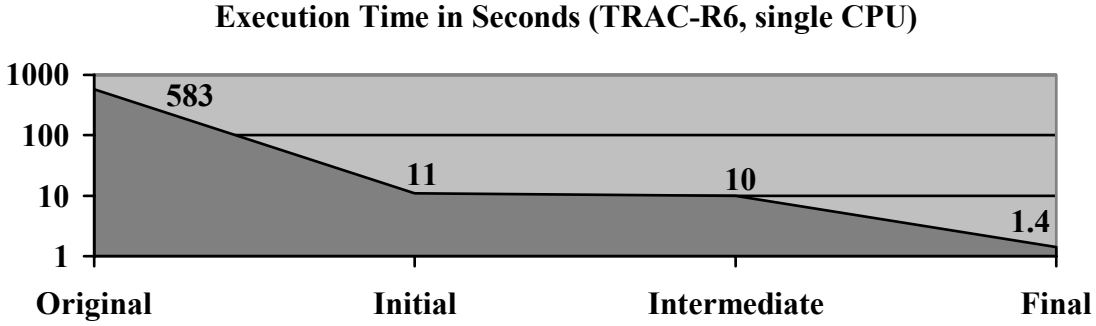


Figure 15 – Current Clustering Performance Improvements

2.1.10 Reference Cluster Processing

Some time was spent investigating performance losses outside the formally declared bottleneck areas and optimized as well. The "reference cluster processing" event is composed of several algorithmic functional steps, which associate and merge new current clusters generated for the current scene with reference clusters from the previous scene. This effectively produces time-smoothed tracks of military units based on the instantaneous noise-laden current cluster samples. The reference cluster-processing step required two operational modes based on the scenario-state. If reference clusters exist from a previous interval, then this step automatically was run in parallel. Otherwise, if this was the first scene and no reference clusters exist, then reference cluster processing was kept on the group leader and is run sequentially. This was implemented due to the fact that the majorities of processing "for-loops" from this step iterate on the reference cluster list and therefore become a no-op when no reference clusters exist. This was too expensive to execute on multiple processing nodes when the majority of time in this step was lost to messaging overhead in the distribution of the current cluster list and the collection of the resulting reference clusters. In this case the algorithms execute on one CPU only.

3 Fusion Architecture Evaluation and Definition

At this point the bottlenecks within the Force Aggregator have been investigated and the reasons for the processing limitations known. The objective of this section is to define an affordable, scalable next generation fusion and exploitation architecture that is capable of supporting the real-time Level 2 assessment. For purposes of this program it is assumed that real-time assessment information needs to be provided in minutes or seconds. Currently, assessment information is on the order of approximately ½ hour to several hours dependent on the scenario that is being processed.

The basic requirements summarized from results from the last section are listed below.

- A large amount of memory is required per processing node. Level-2 Fusion algorithms utilize template-based detection and identification which requires a substantial amount of temporary memory.
- There exists a need for high inter-node bandwidth to transport detailed Level 1 contact reports in and out of the algorithmic steps within the Force Aggregator.
- C++ Support is required. The Evidence Network and Equipment/Organization tables are written completely in object-oriented C++. Conversion to C is possible but developmental costs would be great.

A number of hardware architecture platforms were evaluated and tested to determine their potential as the next generation Level 2 fusion and exploitation architecture platform. Candidate platforms considered were COTS Embedded platforms, adaptive computing platforms, and Linux Beowulf clusters. Major considerations when considering these systems were the cost and availability of the machines at the time of this task. The cost of each solution included the actual cost to purchase the system and support software, and the cost and/or difficulty of modifying the existing Level 2 Fusion system in order to support the proposed architecture solution.

A brief summary of each proposed system follows.

3.1 COTS Embedded Computing Architectures

The Mercury Embedded system was evaluated with great detail, leading to an implementation of the NEAT Force Aggregator on the 16-Node Mercury RACE® system at Integrated Sensors Inc. Mercury has an outstanding record of robustness, reliability, and software support. However this “comes at a price” as this is the most expensive system considered in this hardware trade study. Mercury systems, as well as all other Embedded system types, tend to offer a larger number of processors than other platform architectures and higher inter-node bandwidth but often with the least amount of available memory per processor (128-256Mb maximum today).

The predominant chip technology that Mercury is basing their product offerings today is the PowerPC™ chip, specifically the AltiVec™ series PowerPC processor. The advantage of the AltiVec chip is that it provides both general-purpose processing performance and high-bandwidth data processing for algorithmic intensive computations on a single chip. In order to achieve this capability in previous applications, a two-chip solution was required.

Mercury as with other embedded processing architectures can be used in conjunction with a Sun workstation used as a co-processor. This poses an excellent potential solution for NEAT as the internal Sun workstation could run Sybase and NEAT, and the Force Aggregator would be ported to and executed on the selected embedded hardware. Mercury™ also offers ruggedized systems and has an extensive background in military applications.

However, Mercury as well as other COTS embedded computing architectures are very expensive in both hardware and software products when compared to alternatives.

3.2 COTS Adaptive Computing Architectures

Adaptive computing hardware is generally considered to be field programmable gate array (FPGA) boards. An FPGA processor can be modified at almost any point during their usage to adapt to changing requirements as they evolve. When the application system is upgraded and the algorithms modified, an FPGA can be reprogrammed to meet the evolving needs. This is not a stand-alone computing architecture, but would be used in conjunction with either COTS embedded computing architectures or Beowulf Linux clusters.

The two main chip technology companies that are providing commercial FPGA boards are Xilinx™ and Altera™. Xilinx currently is one of the major providers of FPGA chips that are being used on commercial boards, where each chip is capable of supporting 1-2 million gates. The Annapolis WILDSTAR™ board is an example of a commercial FPGA board that is based on Xilinx parts.

The WILDSTAR board requires the use of the VHDL programming language to define a logical state diagram. While a WILDSTAR emulator was available to speed code development and validation, time spent in software development was costly. Additionally, the Bayesian algorithms within the force aggregator did not lend themselves easily to the FPGA architecture. The WILDSTAR board seems to be more suited for signal and image processing where data sizes are constant and algorithmic operations are predictable and for the most part, repetitive.

3.3 Beowulf Linux Clusters

A low cost candidate solution for the next generation Level 2 exploitation and fusion architecture is the Beowulf class of supercomputers. A Beowulf machine is built from commodity parts such as personal computers and workstations and is connected together using high-speed networking hardware such as Myrinet™ or GigaNet™ to achieve supercomputing speeds. Beowulf actually refers to a special version of the Linux operating system modified to function across multiple computers and allows the user to utilize one large virtual machine. However, a more common alternative is the installation of 'normal' Linux on all processing nodes, and configuration of network and security of the machines to allow easy host-to-host access. The later option, with a GigaNet network connection, is the trial machine for this consideration.

The Beowulf is the lowest cost hardware and software solution, and also offers the highest memory per processing node (512Mb-1Gb). The processing nodes are simple desktop PCs and all software that is required for this system is free. However, the GigaNet network layout (110 Mb/sec maximum) is considerably slower than Mercury's Raceway (267 Mb/sec).

3.4 Trade Space comparison

Figure 16 compares and summarizes the various performance aspects of the proposed solutions.

Computer	Memory Per CPU maximum	Intra-Node Bandwidth	C++ Support	Hardware Costs (per Gflop)	Software Costs	Hardware Maint.	Software Maint.
Mercury	128 Mb	267 Mb/s	Yes **	$6,940 ***	$11000	Low	Low
CSPI	256 Mb	242 Mb/s	Yes	$5,520 ***	$8800	Low	Low
Linux	1 Gb+	110 Mb/s*	Yes	$400 ***	$0	Moderate	Zero
Wildstar	12 Mb	267/242	No	N/A	$15000	Low	Moderate

*Bandwidth based on GigaNet™ back-plane.
**Mercury PowerPC C++ Developer's toolkit available at approx. $10,000 additional cost.
***Average cost per GFLOP across 2, 4, and 6 processing nodes.

Figure 16 – Trade Space Comparison Summary

Due to the Beowulf's overwhelming low cost, availability of memory per node, and performance potential, it was chosen as the hardware architecture solution for the HPRTF system. During development and testing, ISI leveraged an already existing in-house Linux Cluster. A large 64 node+ Linux Cluster in building 106 at the AFRL Rome Research Site will provide an installation and demonstration platform.

4 HP-RTF System Validation

At this point the HPRTF system was prototyped and tested against scripted test cases. The software architecture facilitated a parallel Force Aggregation function but remained stand-alone from the original NEAT architecture. Under this task the HPRTF system was to be validated and verified by integrating HPRTF as the Force Aggregator component into the existing NEAT system. This is in essence returning the divorced Force Aggregator back to the software system it originated from. This was accomplished by using the HPC-link data transport object (see Section 2.1.2 and Figure 5), which had previously been used only in "file" or "offline" mode during software development and testing in Section 2. The socket mode of HPC-link uses the same file descriptor and read/write commands as in disk/file mode, and so was easily adapted to network communication. Additional work was performed on the NEAT host-side to correctly utilize HPC-link for NEAT-to-HPRTF timings and data transfers. This setup is illustrated in Figure 17.

However, it became desirable to write the input cache (Level 1 Intelligence reports) to a file shared across NFS to the HPC machine. This allows modules in HPRTF the capability to read and process the input data cache directly from disk, preferably a disk mounted on the HPC system. This offered a performance improvement as all processing nodes can simultaneously read, parse, and process the intelligence reports without waiting for the group leader to receive and distribute the data from a single TCP socket over a standard LAN.

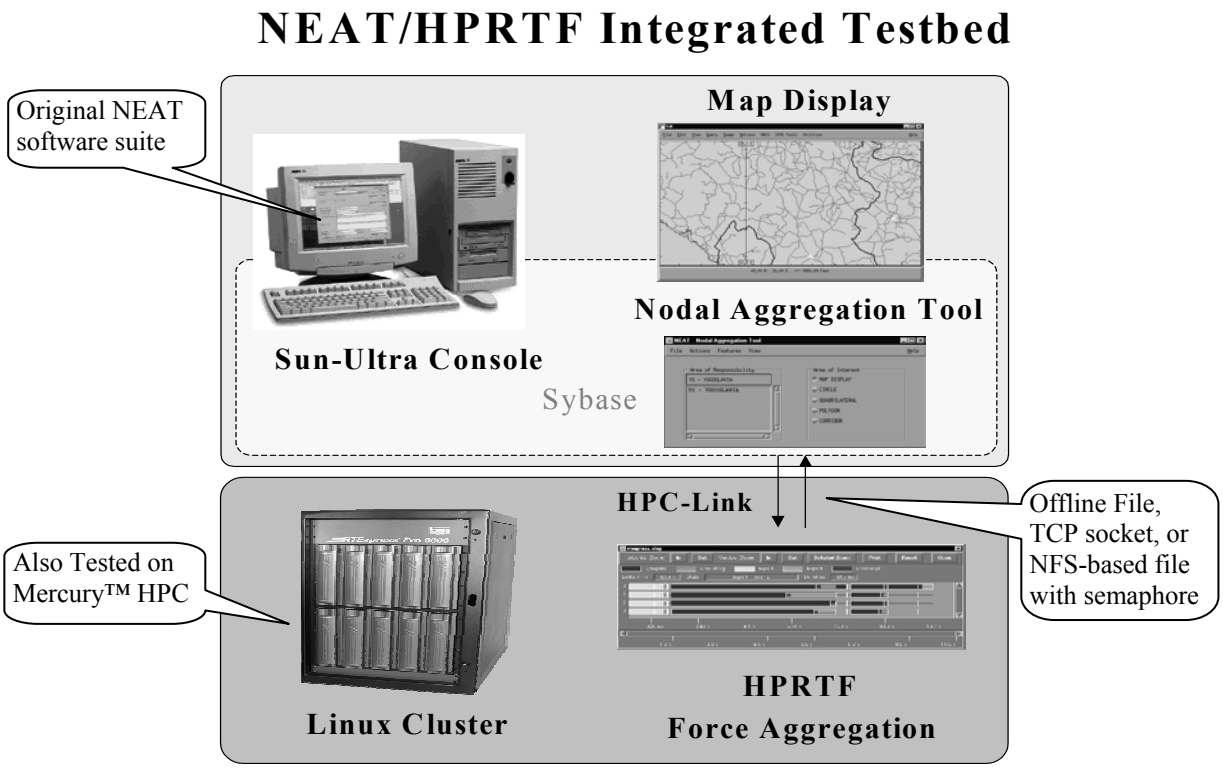


Figure 17 – NEAT/HPRTF Integrated Testbed

5 Demonstrations and Results

All original tasks of the HPRTF effort were completed. Processing bottlenecks that limited NEAT from performing in real-time were identified. The major performance bottlenecks included Sybase access, the Evidence Network Algorithm, the Contact Association Computation, and the Current Clustering Algorithm. After which, a development of a hardware requirements trade-space comparison was made to identify affordable solutions. Linux was chosen as an ideal implementation platform, but the code was also built and executed on Mercury and Sun platforms to test portability.

The fusion architecture was then prototyped using RTExpress and Matlab tools. HPC-link and a robust Level-2 data management library formed the HPTRF backbone. Algorithms from NEAT were wrappered in HPRTF, parallelized, and tested. Many software development iterations occurred to test different implementation strategies and perfect the system to achieve maximum performance.

The HPRTF system was then integrated with the original NEAT tool suite to validate the system as a whole. Minimal time and effort was spent on this effort as the GUI front-end to NEAT was reused. This allowed the use of the original NEAT operator interface to drive the HPRTF fusion engine, as well as the many other NEAT components which read and write from the Sybase database.

Demonstrations of working HPRTF prototypes were held at both ISI and the Rome Research site. The sequential Force Aggregator was demonstrated on Sun, Mercury, and Linux platforms during a status meeting 20-Nov-01 at ISI. The purpose of the demonstration was to show portability of HPRTF and RTExpress between hardware architectures. This consisted of HPRTF running on a single processing node on an HPC and the NEAT host software running on a Sun Sparc.

A second demo occurred at 10-Apr-01 where the parallel Force Aggregator was run on a 4-node Linux cluster, connected to the NEAT host GUI running on a Sun Sparc. In addition, a side-by-side comparison was demonstrated with both the original NEAT software build and the integrated HPRTF version running on the Linux cluster. Figure 18 depicts the demonstration layout.

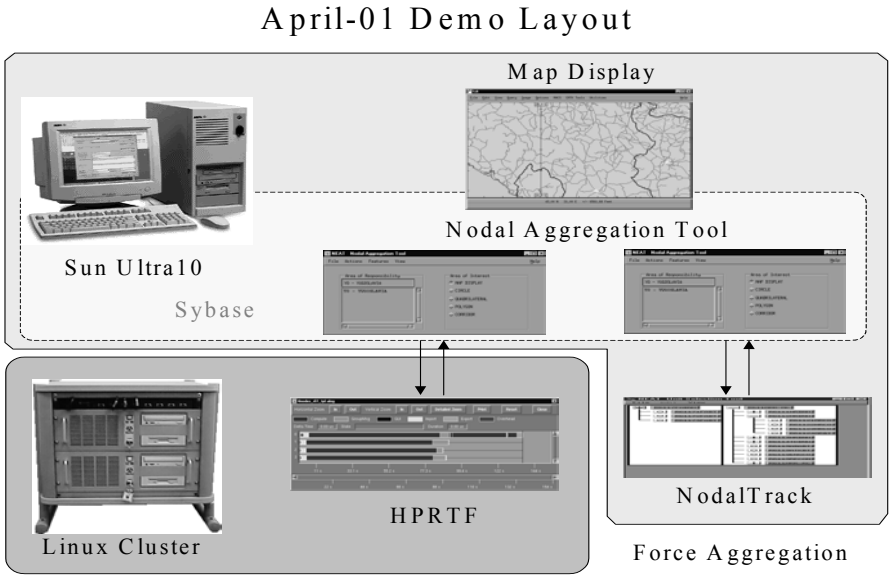


Figure 18 – April-01 Demonstration Layout

A third demo occurred at 4-Sep-01 where the parallel Force Aggregator was run on 5 nodes on the HADES Linux Cluster located in building 106 at the AFRL Rome Research Site. NEAT was installed onto a neighboring Sun Sparc workstation to replicate the appearance of the previous April-01 demonstration. The three test-case demonstration scenarios, SAM-618, TRAC_R6, and SCEN-R61 (see Section 1.4 for scenario descriptions), were demonstrated using the NEAT GUI on the Sun host while running the Force Aggregation process on the Linux Cluster. In addition, a side-by-side comparison of the Force Aggregators was demonstrated with both the original NEAT software build and the integrated HPRTF version running on the Linux cluster.

Performance results from the HPRTF project met the original requirement and goal of real-time Level-2 Fusion. "Real-time" in this context is the ability to provide JDL Level-2 situation assessment in minutes or seconds. Where the original software/hardware required hours to run, the HPRTF version ran in minutes. In cases where the original product required minutes to run, the HPRTF version functioned in seconds. A summary of performance improvements is shown in Figure 19.

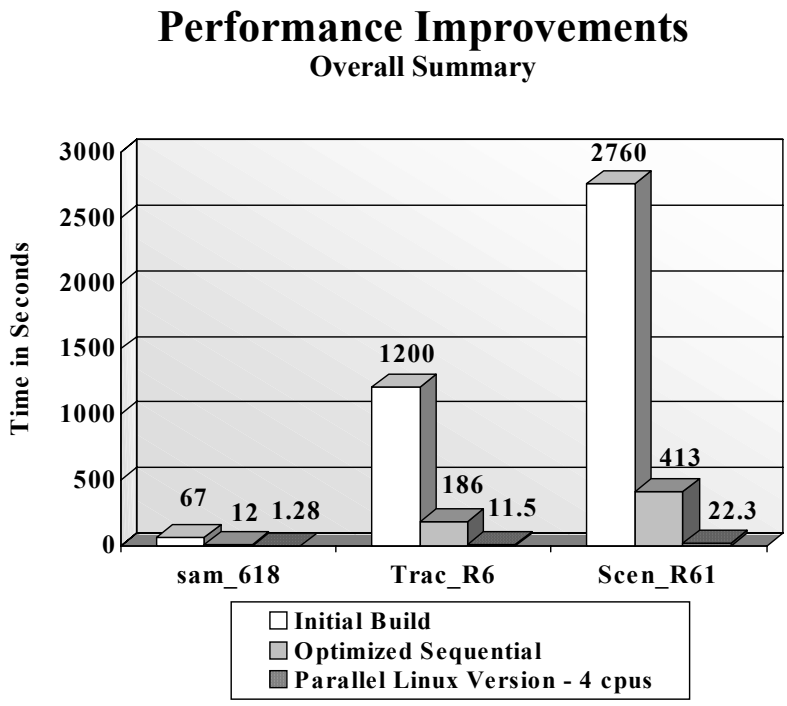


Figure 19 – Performance Improvements Summary

Performance improvements were not limited to parallelizing existing algorithms across multiple processors. The sequential algorithmic processing was optimized as well, keeping in mind parallelization of a non-optimized function does not yield the best results. Figure 20 illustrates the scalability of the HPRTF implementation. This is followed by Figure 21 that offers a performance comparison between the final results and the initial Linux software build. This comparison reflects both sequential and parallel improvements combined. In all trials the software was executed on 600Mhz PC CPUs.

Compared to the original Force Aggregator, these final results show an average speed improvement of 47x when HPRTF is executed on one CPU, and 95x average improvement when run on 4 CPUs. This means that the Force Aggregator ran an average of 47 times faster with sequential optimizations alone, and averaged 95 times faster when run in parallel on 4 CPUs. Better gains in performance were made possible in the parallel version by first streamlining processor and memory usage in the sequential version. This demonstrates the importance of both sequential and parallel optimizations to improve speed performance.

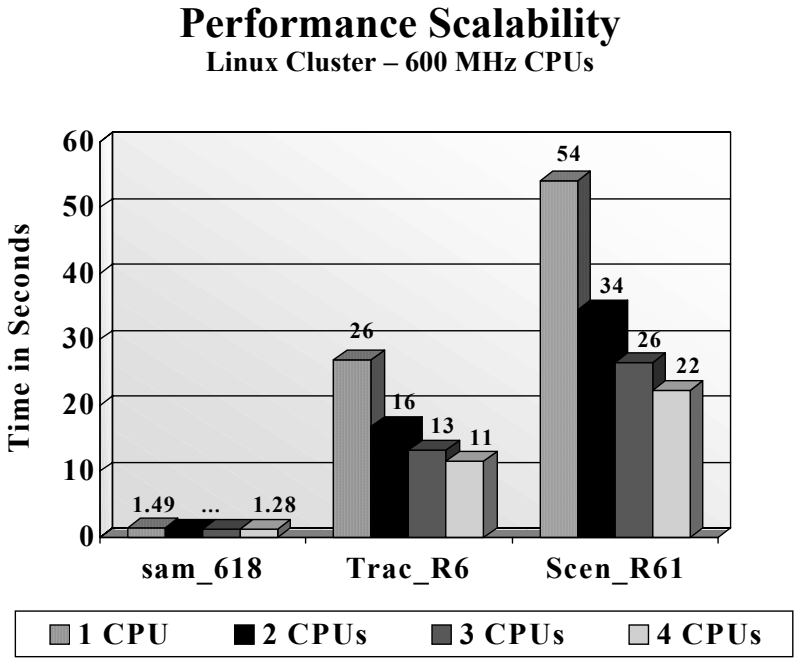


Figure 20 – Performance Scalability

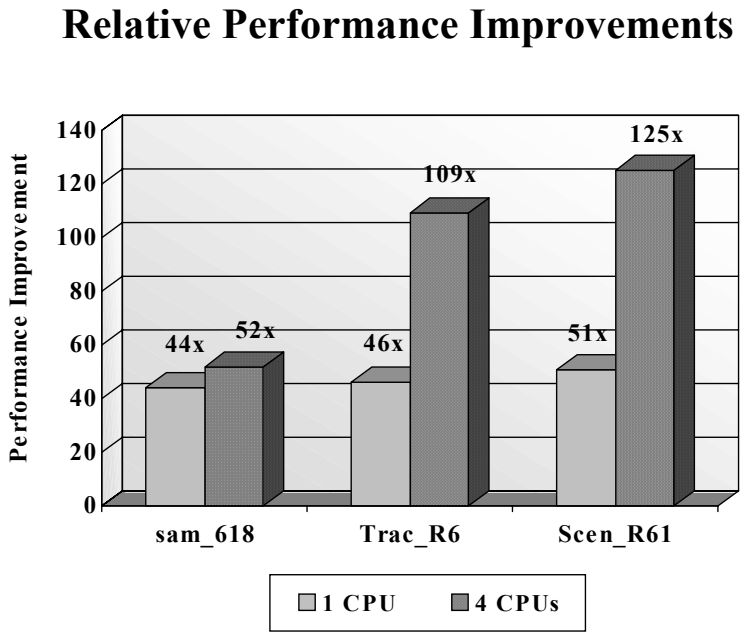


Figure 21 – Relative Performance Improvement vs. Initial Build

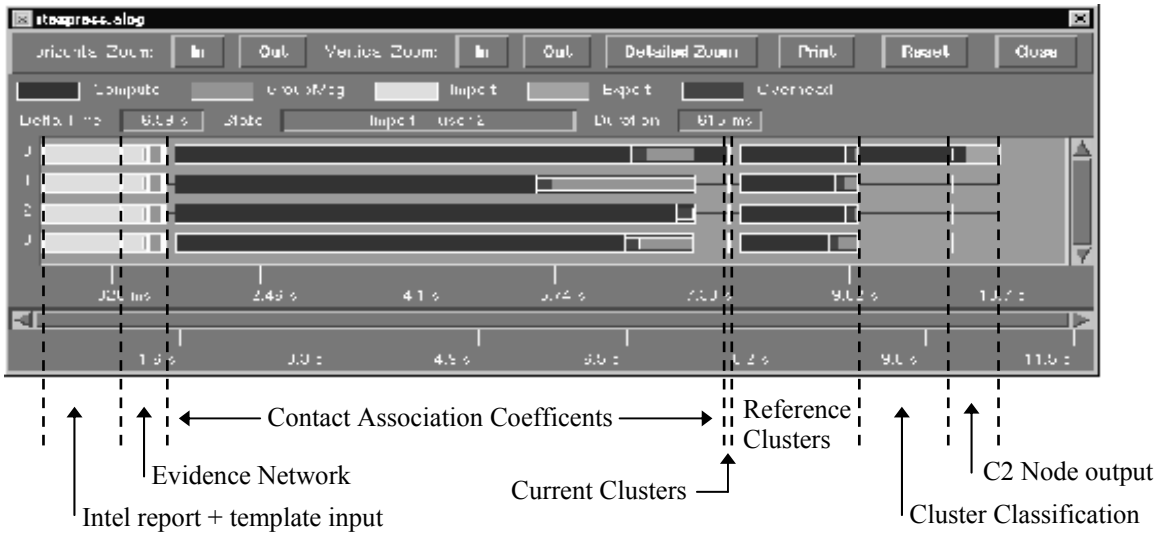


Figure 22 – RTExpress Visualization

Figure 22 above shows an RTExpress display which depicts what each processor is doing and for how long, and gives insights to timing issues across multiple nodes. Shown is the parallel Force Aggregator processing the Trac_R6 scenario running across four 600 Mhz CPUs. Execution time runs from left to right. Colored sections are processing intervals and gaps indicate idle or a wait condition. The highlighted bands near the end of the Contact Association Coefficients and Reference Clusters indicate inter-processor communication, or the passing of data between CPUs.

Figure 23 illustrates RTExpress visualization of multiple runs using an increasing number of CPUs. Depicted is the parallel Force Aggregator processing the Trac_R6 scenario. To the left are graphical representations of what each CPU was doing during the run. See Figure 22 for details concerning each of the RTExpress windows. The graph to the right is a composite report of the results from each run. Shown here is the amount of time spent in each Force Aggregator algorithm versus number of CPUs used. From this one can easily see the performance improvement gained from each additional CPU. The diminishing gain as more processors are added is common in the field of parallelization and is due to increasing time in messaging between CPUs.

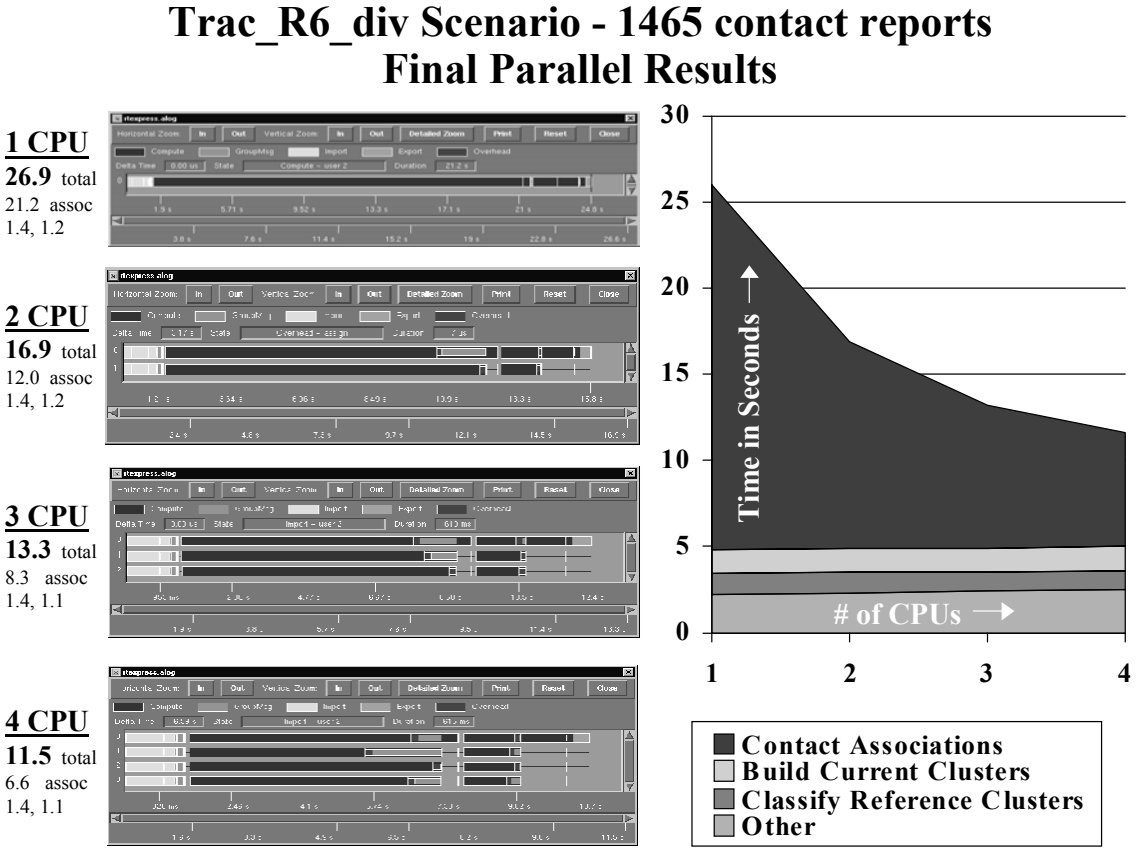


Figure 23 – RTExpress Visualization, 1-4 CPUs

6 Conclusions

The HPRTF program was able to define, prototype, and demonstrate an affordable next-generation real-time fusion and exploitation architecture that supports JDL Level 2 fusion. The developed architecture focused on requirements of JDL Level 2 systems, and alleviated the performance bottlenecks that limited NEAT from providing real-time fusion assessment. The parallel and sequential optimizations were accomplished by using RTExpress as a development and diagnostic tool. Compute-intensive algorithmic modules were identified and parallelized. Then using RTExpress visualization tools and the understanding of the algorithmic steps within the Force Aggregator, sequential optimizations were applied to improve performance further. The prototype was installed in building 106 at the Rome Research Site, Rome, NY, and demonstrated at the final meeting. A development environment was also installed to facilitate integration and reuse of the HPRTF products with other projects after completion of the HPRTF contract.

It is hoped that this product either bundled with NEAT as a parallel real-time force aggregator or as the developmental fusion architecture alone, is reused to the fullest extent possible. One possible follow-on effort is the wrappering and interfacing of HPRTF/NEAT as a JBI Fuselet, where the system would act as a real-time Level-2 Fusion agent. This fusion outlet could subscribe to Level-1 intelligence streams, compute the battlefield situation assessment, and broadcast military units at their estimated locations in real-time. There is no other system available that can provide this function in real-time.

7 List of symbols and abbreviations

C2 nodes	Command and Control Nodes or military unit clusters
COMINT	Communications Intelligence
Contact Report	An intelligence report equivalent to a JDL Level-1 track with ID
COTS	Commercial off-the-shelf technology
CPU	Central processing unit, also equivalent to "processing node"
Data Parallel	A parallelization technique where algorithmic processes remain relatively unchanged but the data it processes is partitioned across processing nodes
Force Aggregation	JDL Level-2 Fusion, or situation refinement of Level-1 reports into aggregate clusters
FPGA	Field programmable gate arrays
GigaNet	A 3rd party PCI-based network hardware and software package
Group leader	Lead processing node or CPU within a multi-processor machine
HPC	High Performance Computer, generic for multi-processor computers
HPC-link	File/Socket software interface between NEAT and HPRTF
HPRTF	High Performance Real Time Fusion Architecture
JDL	Joint Directors of Laboratories
LAN	Local Area Network
MTI	Moving Target Indicator
MxArray	Matlab™ generic storage structure type for all data types
Myrinet™	A 3rd party parallel-based network hardware and software package
NEAT	Nodal Exploitation and Analysis Tools – a JDL Level-2 fusion system
NFS	Network File System
Processing node	Single logical node in a multi-node computer. Equivalent to "CPU"
Raceway®	Mercury™ high-speed inter-node communication network
RTExpress	"Real-Time-Express" development software used to build and debug source code across multiple HPC hardware architectures
SAM	Surface-to-Air Missile, or the battery that supports it
Semaphore	Software-level signal to key processes on the same or different machine
SIGINT	Signal Intelligence, specifically those emitted by radars
Template	A single military unit template describing equipment and organization

8 Appendix A - Hardware Trade Study

This appendix presents a study and comparison of available high-performance computing architectures for the implementation of the HPRTF system. There were three systems compared across seven attributes.

The systems compared were:

1. Mercury
2. CSPI
3. Linux Clusters

The attributes investigated were:

a. processing power
b. hardware cost
c. software cost
d. space and weight requirements
e. air flow requirements
f. power requirements

The following pages will detail this study.

8.1 Mercury

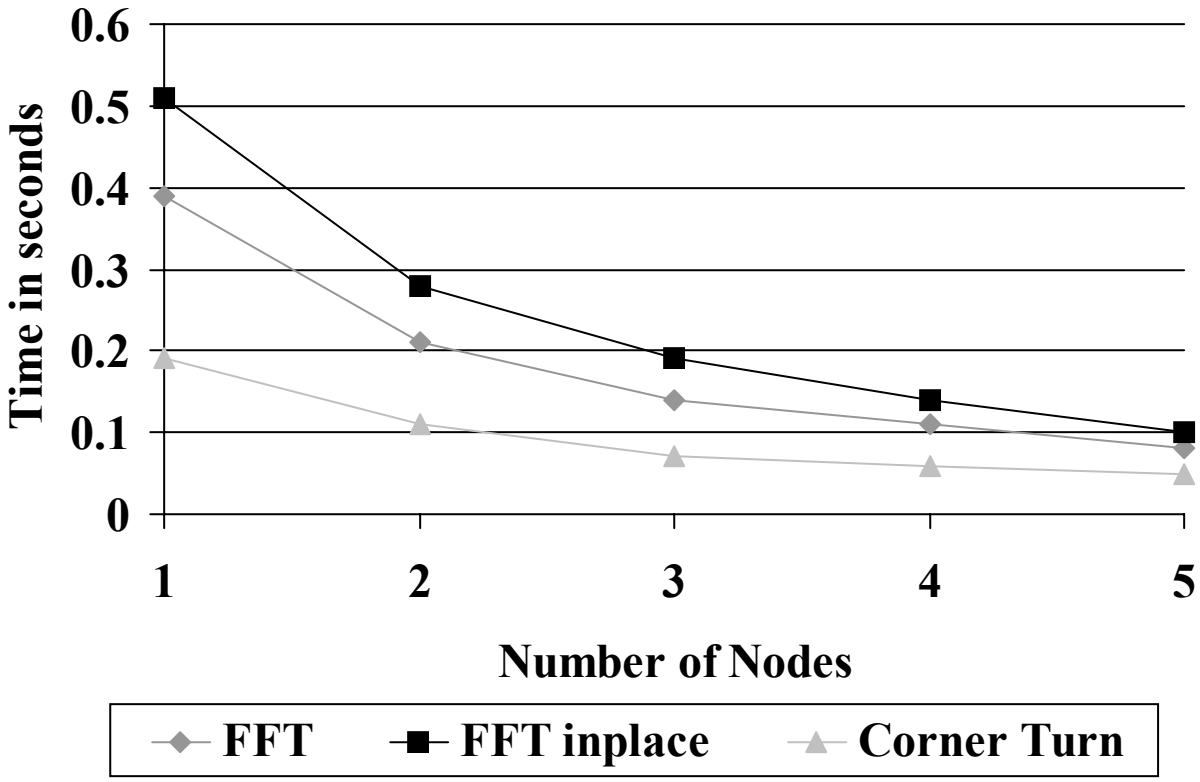


Figure 24 - Mecury Nodes vs. Processing

Base System Requirements (2001 prices)

Chasis (21 Slot 6U VME)	$42,300
Motherboard (MCJ6-VH)	$4,800
C Development and Runtime	$11,000
RACEWAY Interlink	$2,300
Daughtercards	$17,600 each (P2J128J-Q-VH)

2 nodes with (1400Mhz PowerPC, 128Mb RAM) each Race++

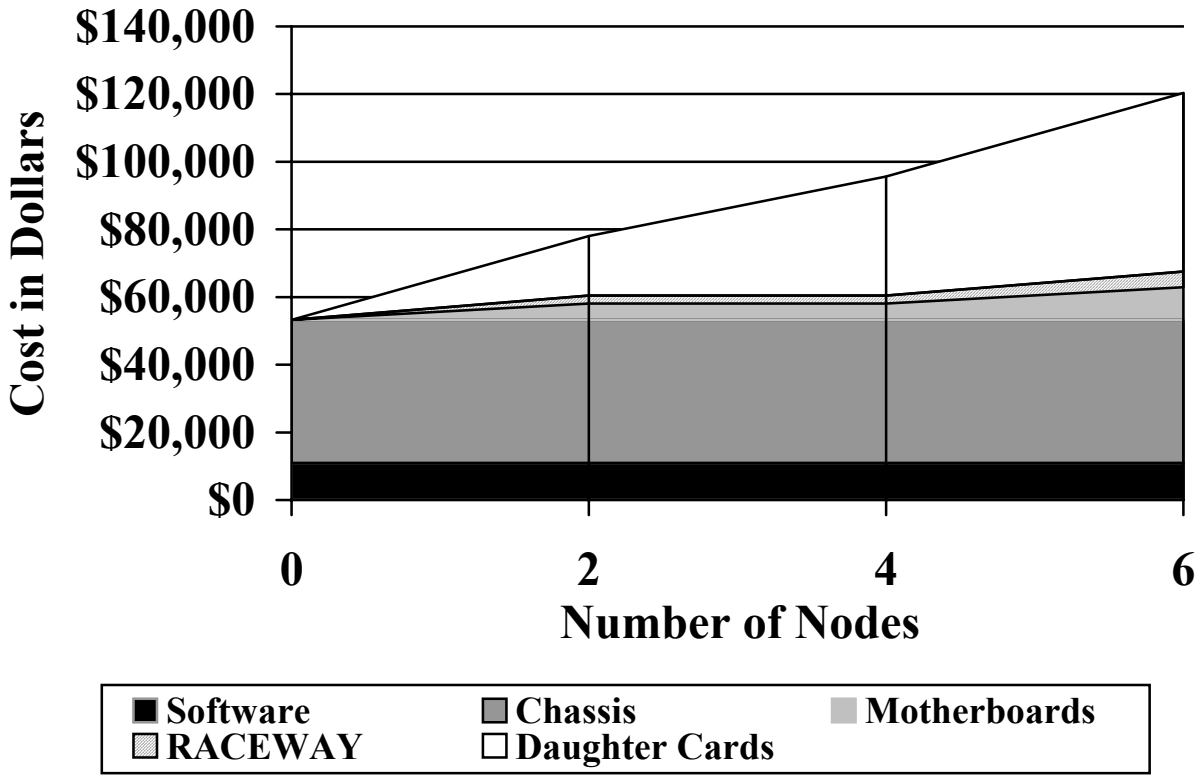


Figure 25 – Mercury Nodes vs. Cost

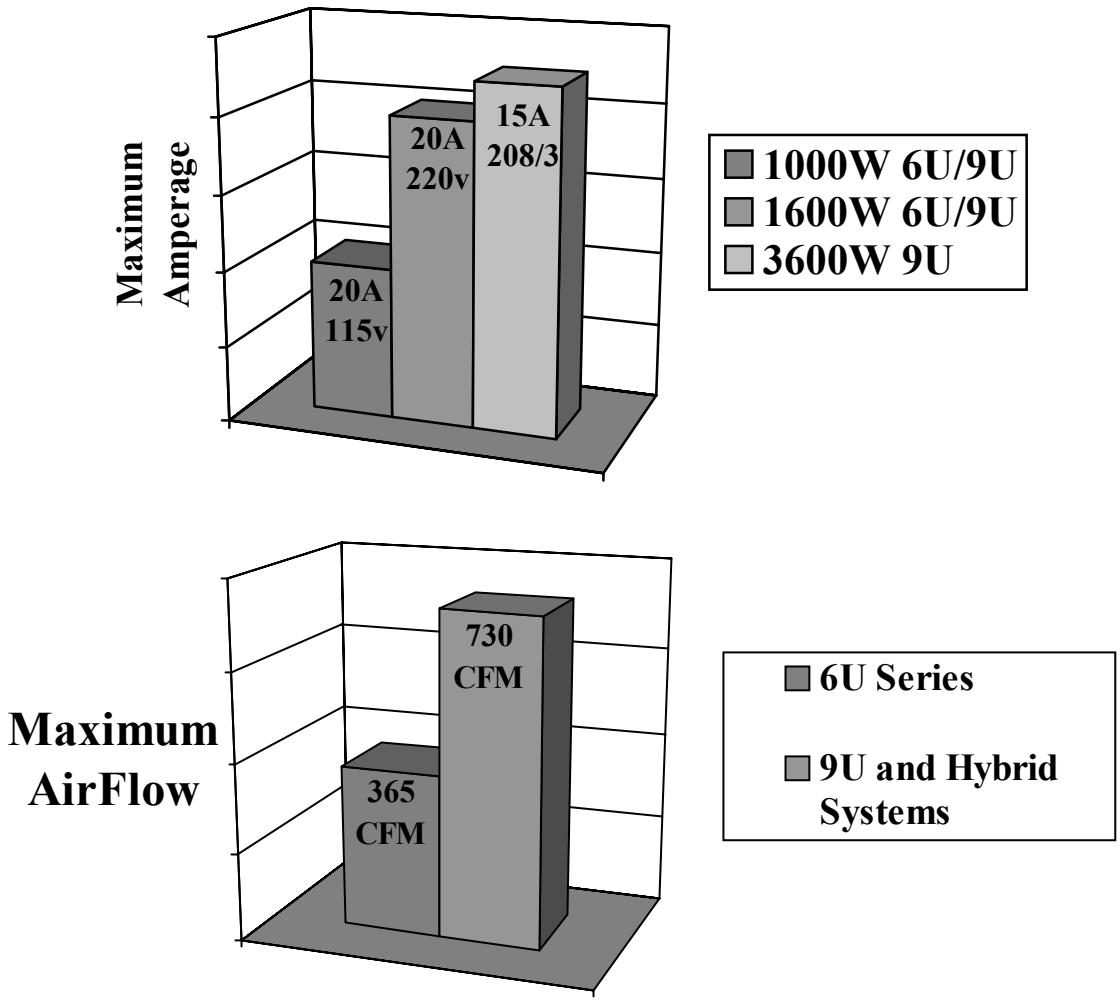
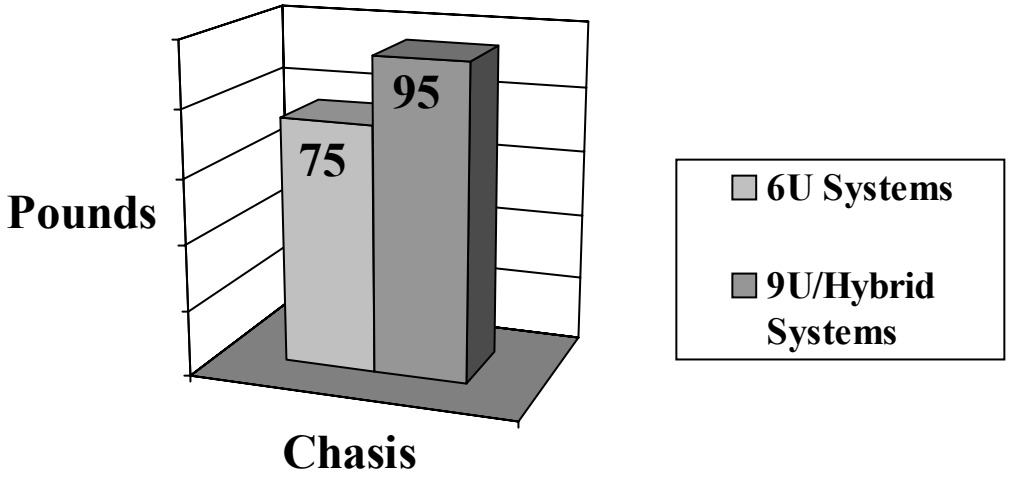


Figure 26 – Mercury Power and Air-Flow Requirements



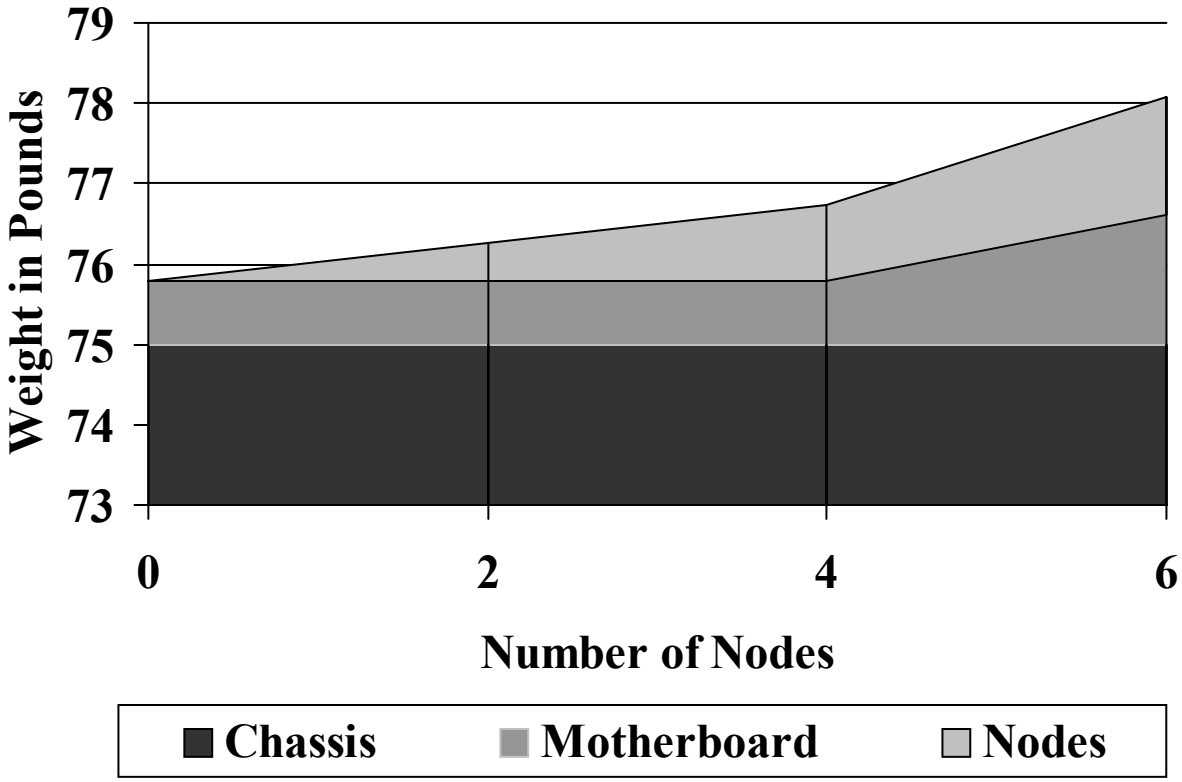


Figure 27 – Mercury Weight

MERCURY CHASIS	6U Series	9U and Hybrid Systems
Dimensions (HxWxD)	17.5x19x21	22.75x19x28
Weight	75 lbs	95 lbs
Air flow	2x365 CFM fans	2x365 CFM fans
3.3 volts	120a	160a
Input voltage	100-120 VAC, or 200-240 VAC, or 208-3 phase	
Frequency	50-60 Hz	
Temperature	5-40 deg C	
Altitude	8000 f t	
Noise	65 dBA	

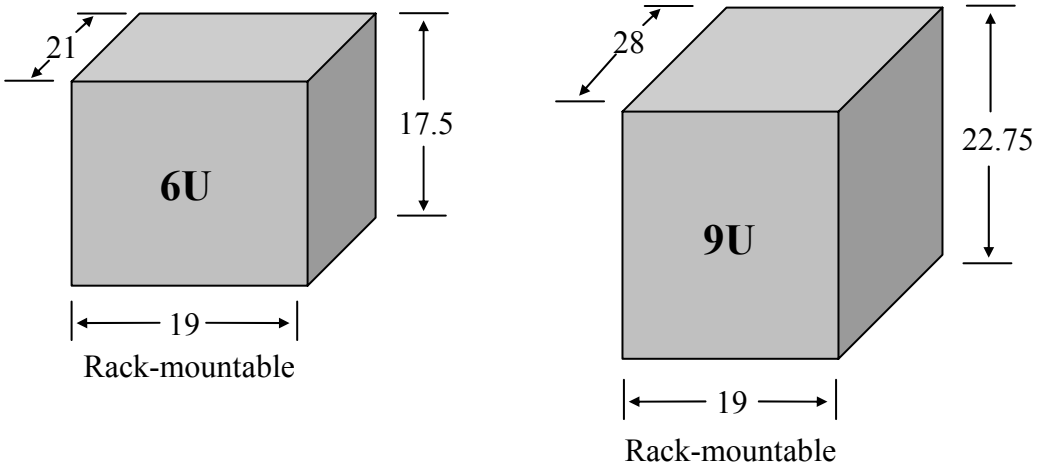
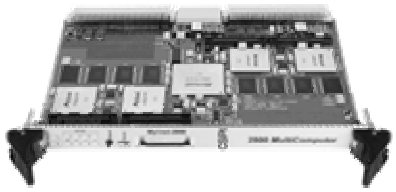


Figure 28 – Mercury Space Requirements

8.2 CSPI



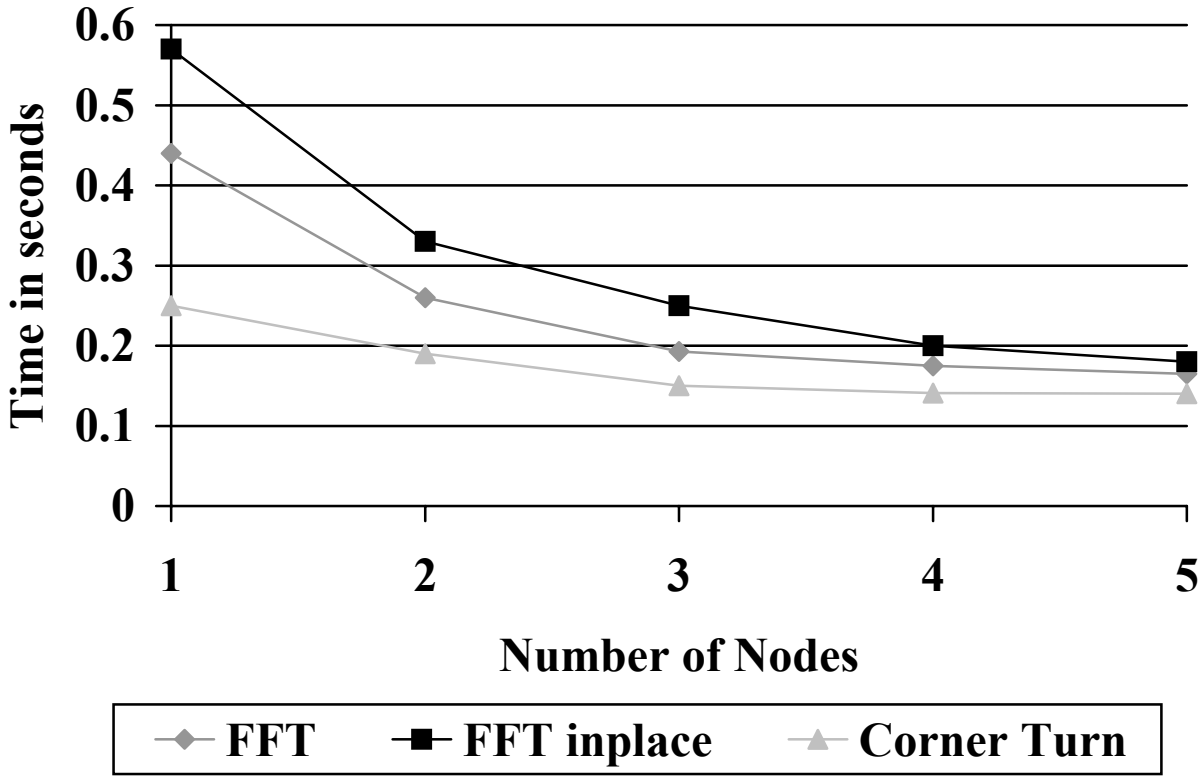


Figure 29 – CSPI Nodes vs. Processing

Basic System Requirements (2001 prices)
Chassis (6 slot VME_) $16,650
Development and Runtime $8,800
M2K Backplace overlay $5,100
Processor Boards $50,906
CSPI 2841, 4x400Mhz PowerPC, 14 Gflops)

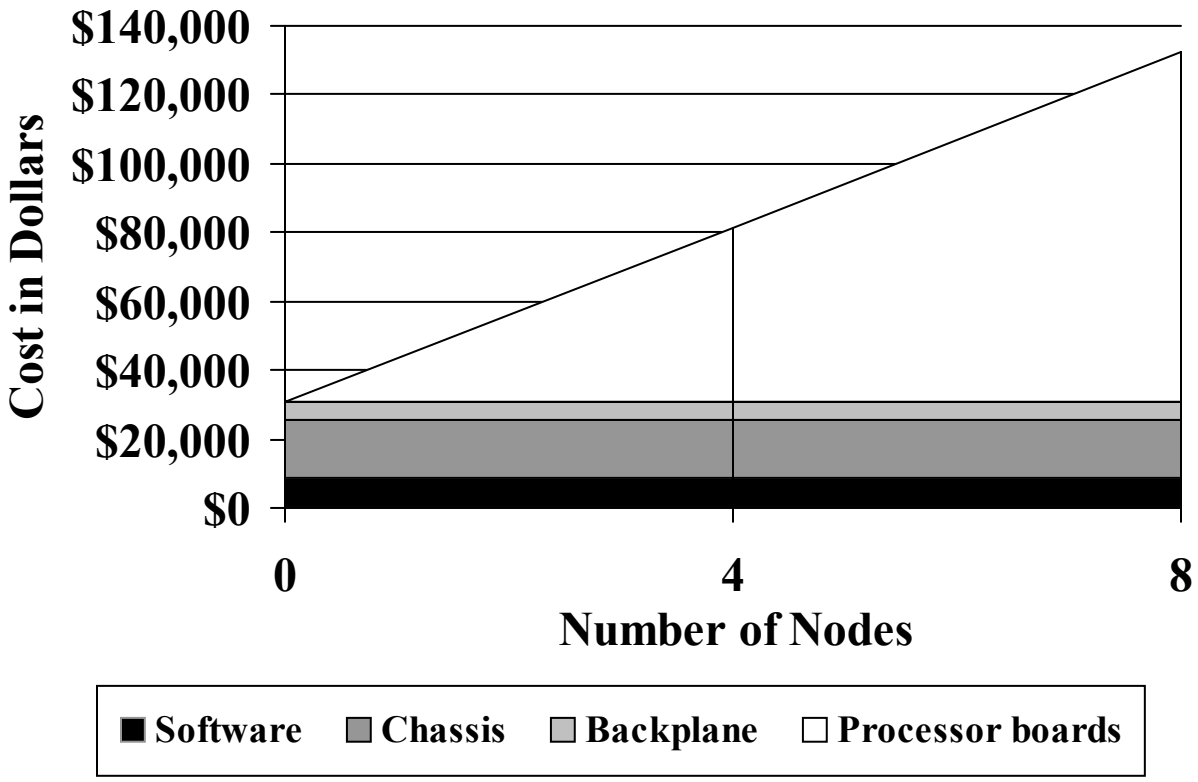
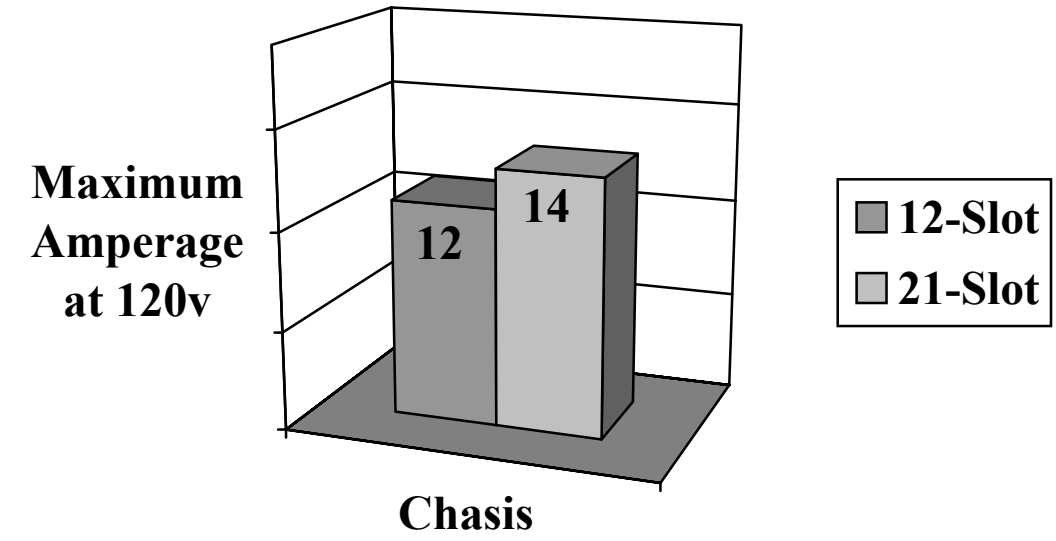


Figure 30 – CSPI Nodes vs. Cost



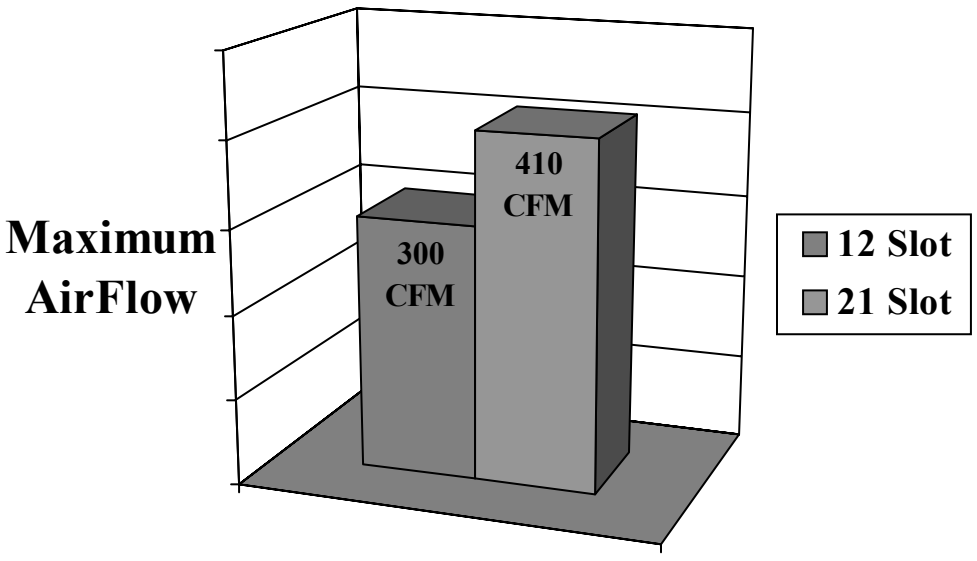
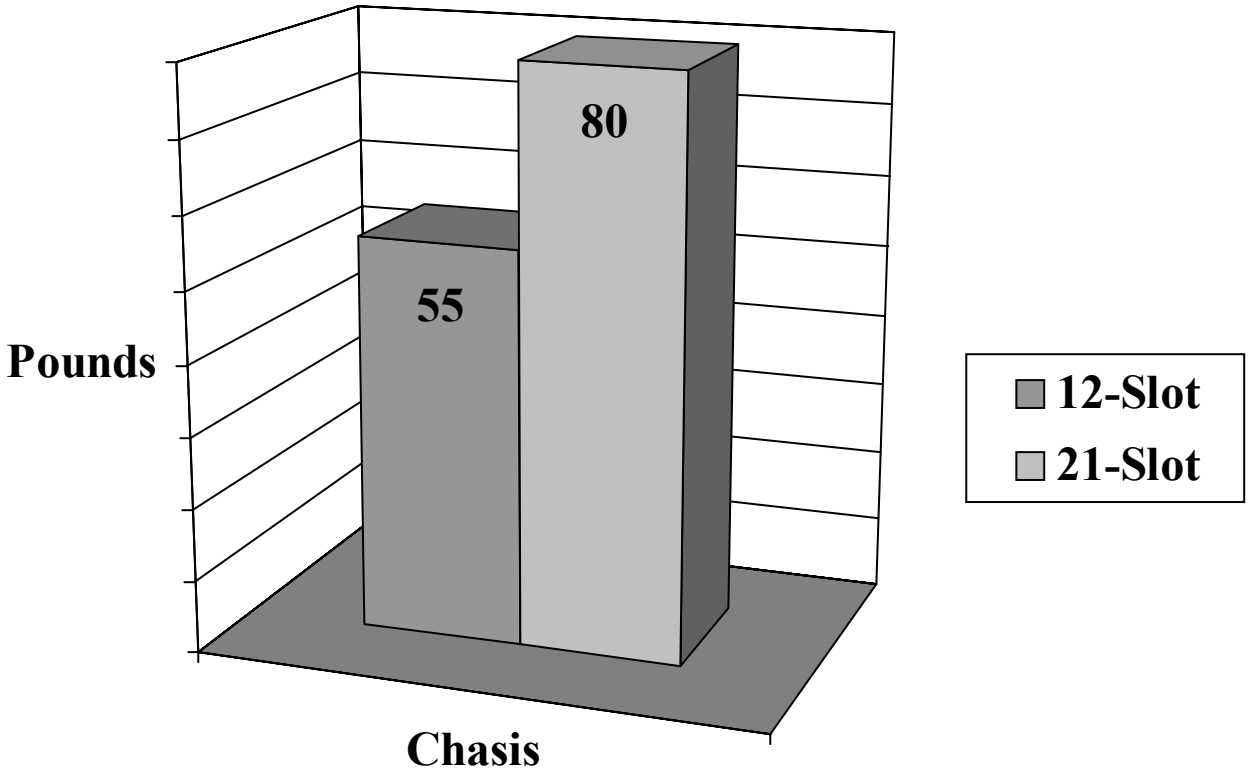


Figure 31 – CSPI Power and Air-Flow Requirements



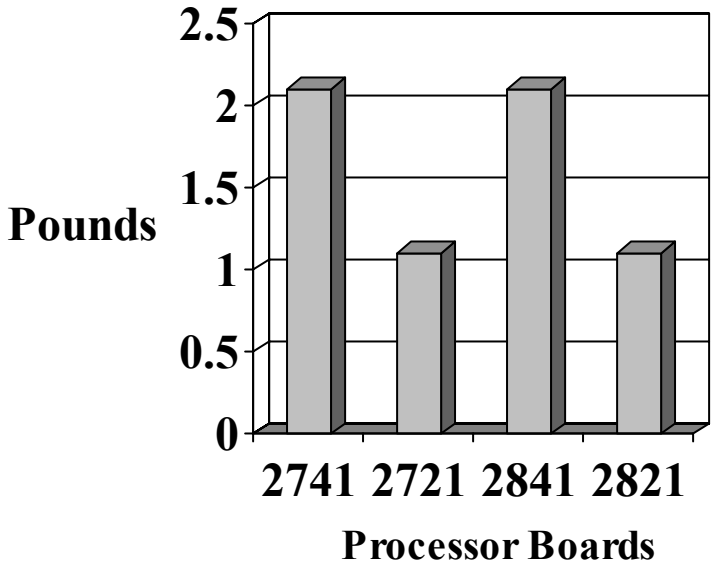


Figure 32 – CSPI Weight

CSPI				
	2741/2721		2841/2821	
	4 cpus	2 cpus	4 cpus	2 cpus
Processors	400Mhz PowerPC 750		400Mhz MPC7400 w/AltiVec	
Memory	256 Mb	128 Mb	1 Gb max	512 Mb max
Node Bandwidth	Myrinet on VME		Myrinet on VME	
Advertised Speed	3.2 GFLOPS	1.6 GFLOPS	14 GFLOPS	7 GFLOPS
Weight	2 lbs	1 lb	2 lbs	1 lb
Temp Range	0-50 deg C at 12 CFM		0-50 deg C at 12 CFM	
Altitude	<10000ft		<10000ft	
Humidity	5-90%			
Power	45.2 Watts typical, 62.0 watts max		25 watts	14 watts
Power 3.3v	25	14	26	14.5
Power 5.0v	30	15	36	18
CHASIS	12-Slot	21-Slot		
5 volts	120a	120a		
12 volts	8a	8a		
minus 12	8a	8a		
3.3 volts	120a	160a		
Input voltage	90-264 volts			
Input Current (max)	12a @ 120vac	14a @ 120vac		
Frequency	47-63 Hz			
Size (HxWxD)	14x19x20.5	17.5x21x19		
Weight	55 lbs	80 lbs		
Cooling/Air flow	3x100CFM fans	1x410 CFM fan		
Temperature	0-40 deg C			
Humidity	10-95%			

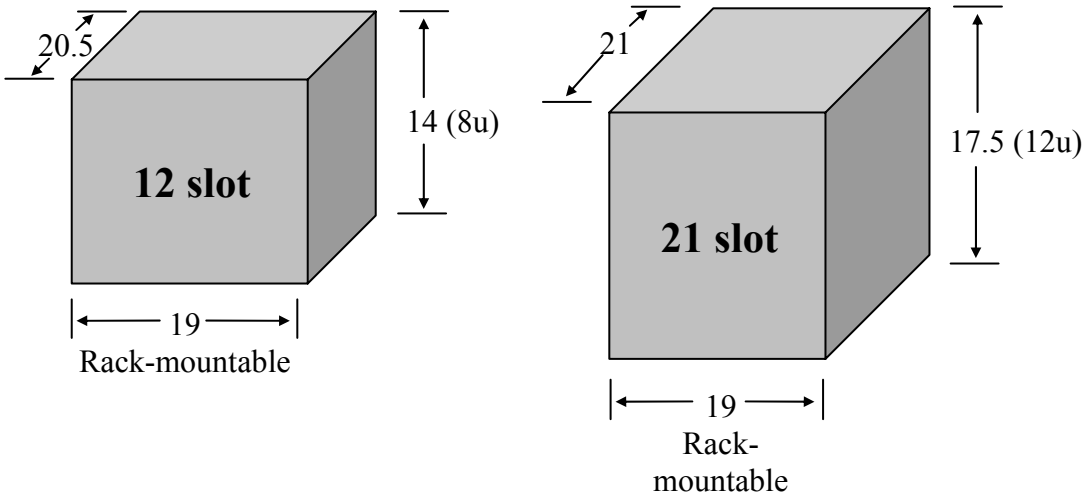


Figure 33 – CSPI Space Requirements

8.3 Linux Cluster

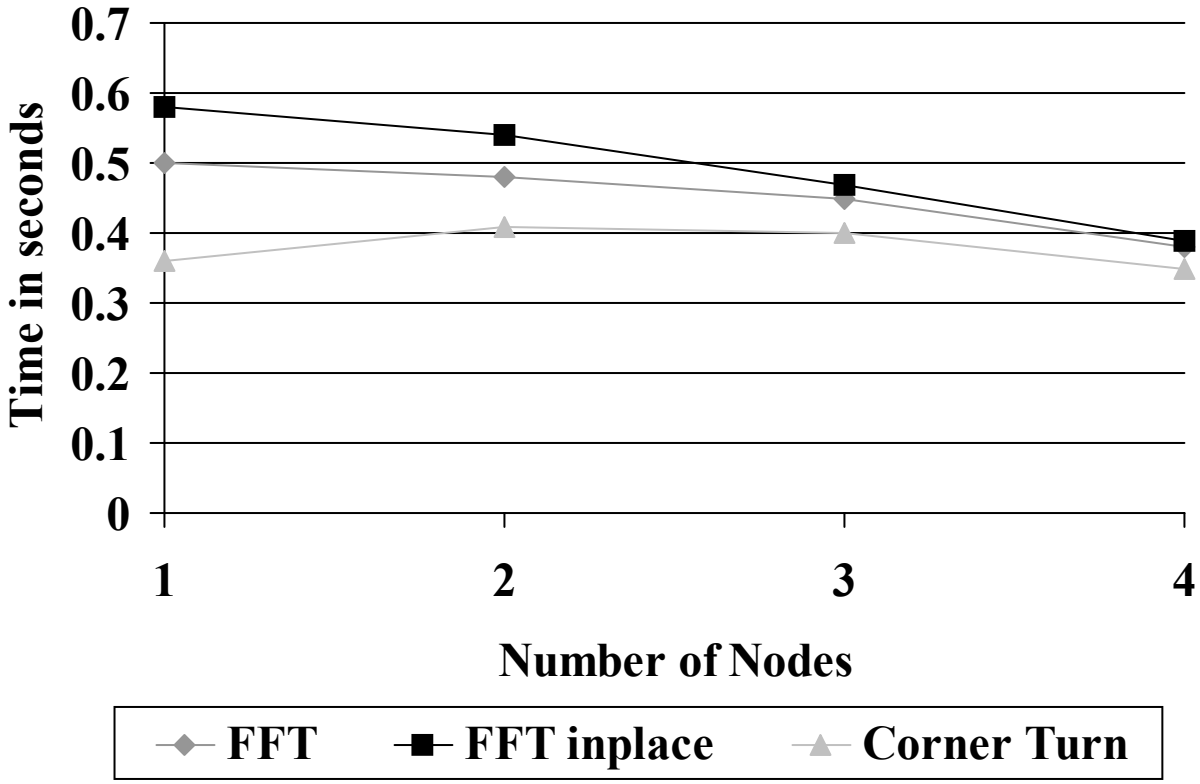


Figure 34 – Linux Cluster Nodes vs. Processing

Base System Requirements (2001 prices)	
GigaNet Hub	$2500
All Software	Free
AMD 1.2 Ghz 512Mb RAM node	$210
GigaNet Network Adapter	$450
1U Rack Case w/150W PS	$180

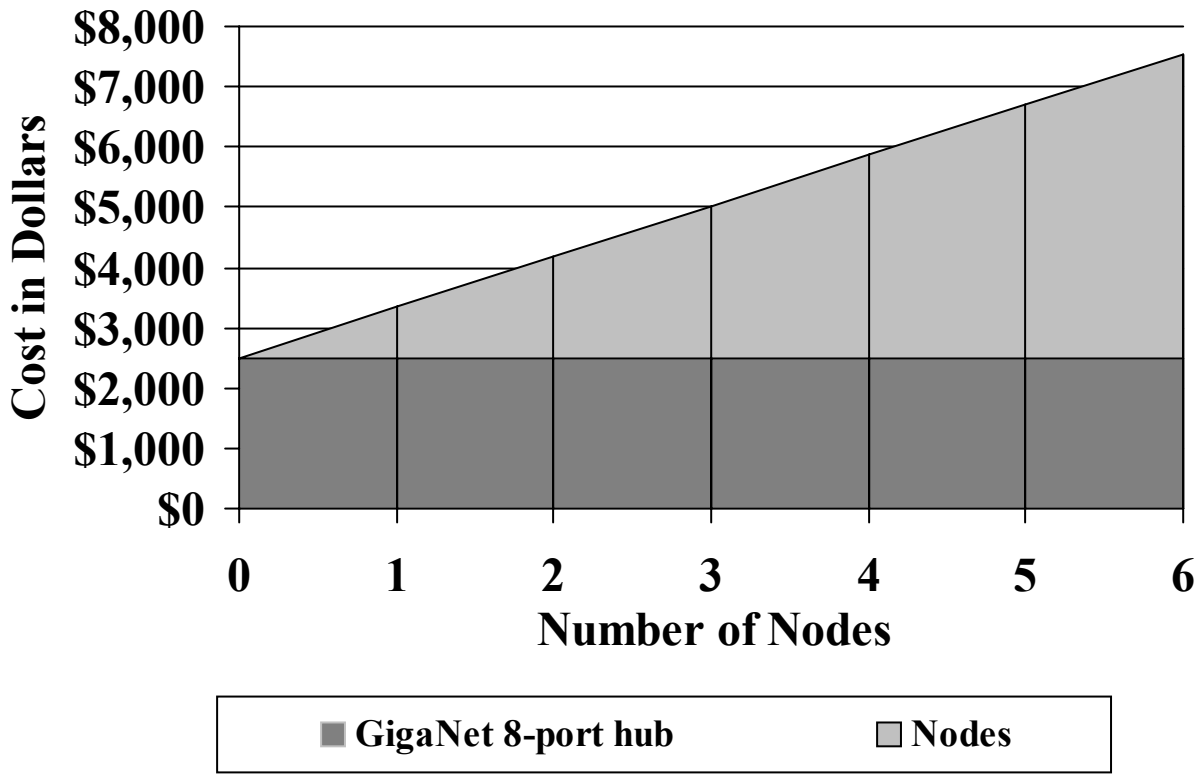
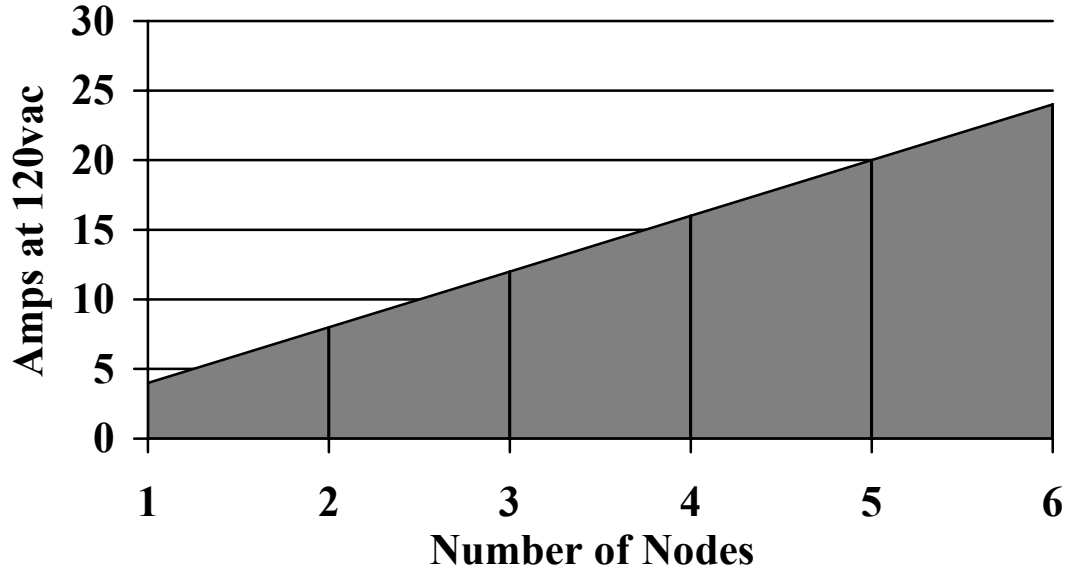


Figure 35 – Linux Cluster Nodes vs. Cost



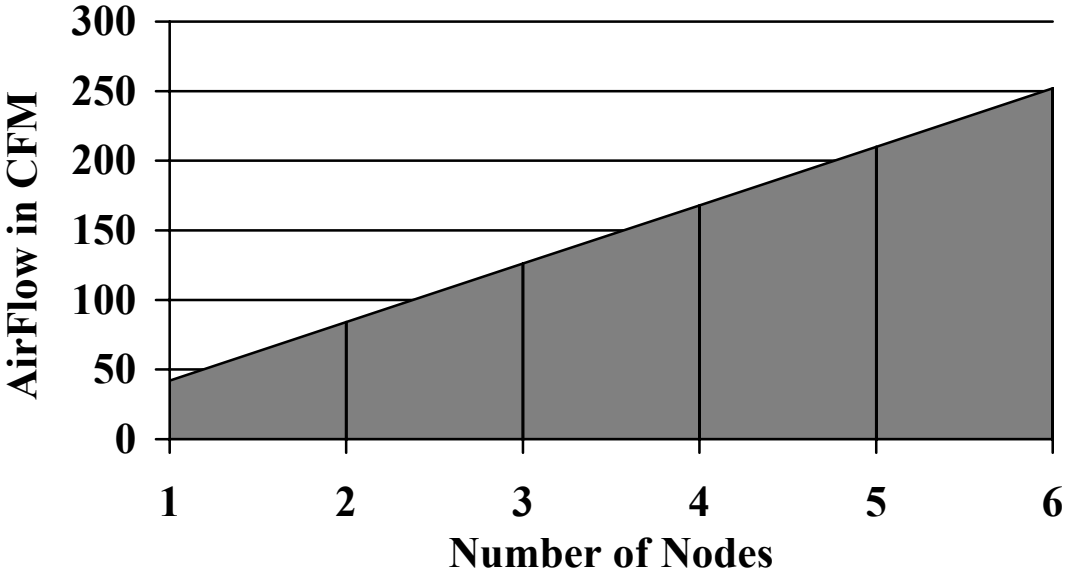
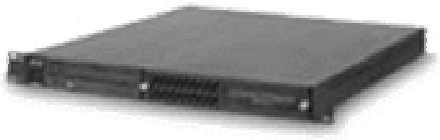


Figure 36 – Linux Cluster Power and Air-Flow Requirements



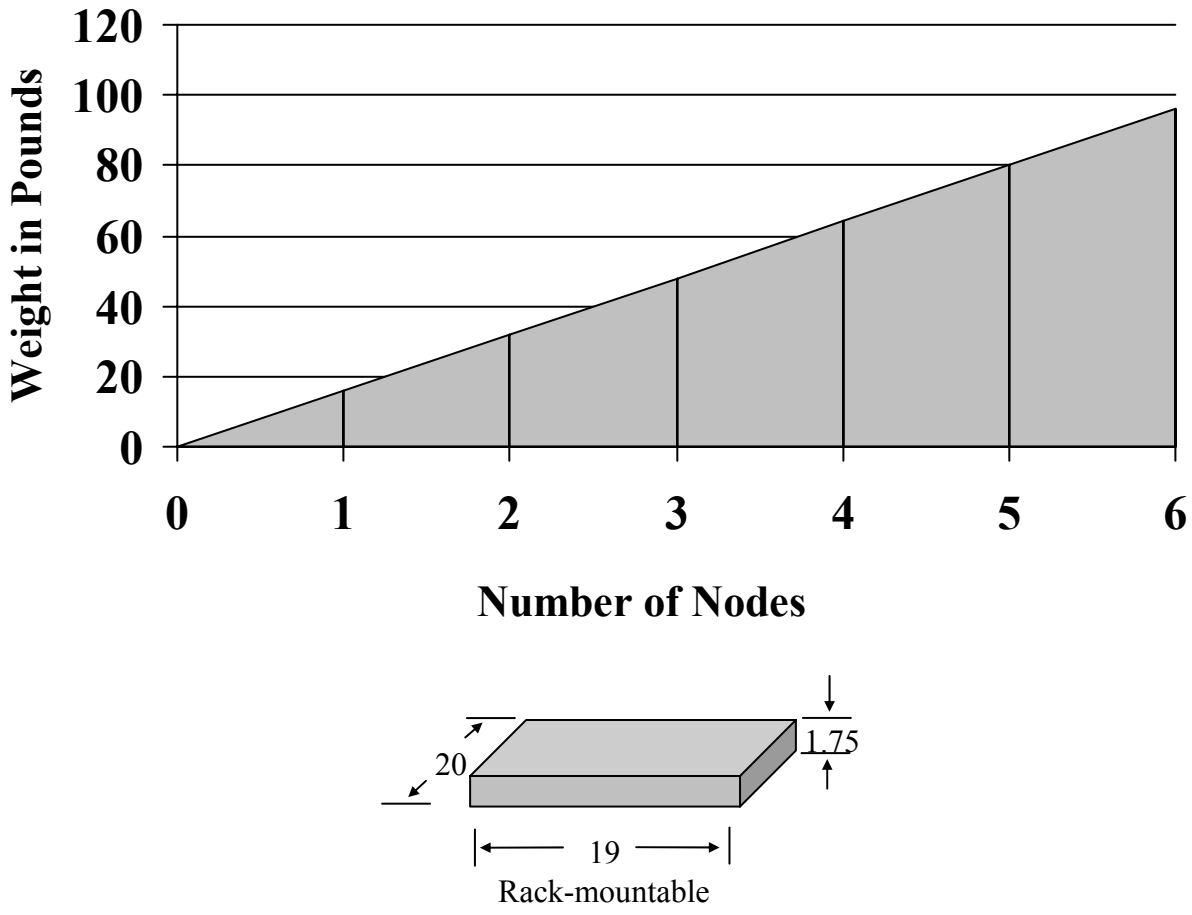


Figure 37 – Linux Cluster Weight and Space Requirements

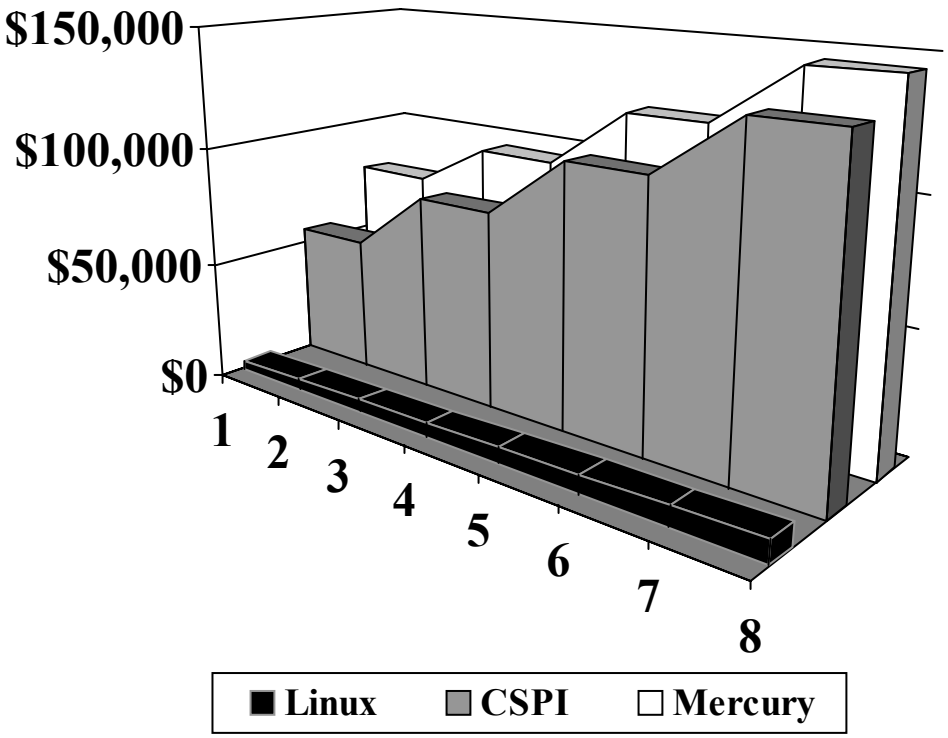


Figure 38 – Cost Comparison

9 Appendix B - Users Guide

Installation

The HPRTF software is written in C/C++ and compiles using RTExpress to build a parallel version of the Force Aggregator component from NEAT. RTExpress must be installed and configured on the target machine before compiling the HPRTF software. This installation procedure assumes the target host runs Linux, however the code has been previously compiled on Sun and Mercury platforms. C++ support is required, and GCC version 2.95.2 or higher is suggested.

Contact Integrated Sensors Incorporated for questions and availability of RTExpress software.

Integrated Sensors, Inc.
502 Court St., Suite 210
Utica, NY 13502
(315) 798-1377

The HPRTF application, once built, can run standalone from data files or in concert with a special modified version of NEAT in a demonstration mode. If the demo mode is desired, it is strongly recommended that a "neat" home account be installed on the target HPC machine, and the NEAT host (requires a Sun with Solaris 2.6 or higher) share the account via the "auto-home" Unix mechanism. In this manor, data files can be easily shared across machines and no HPC processing latency is introduced as the account physically exists on the HPC side. Consult the system administrator of the machines if necessary.

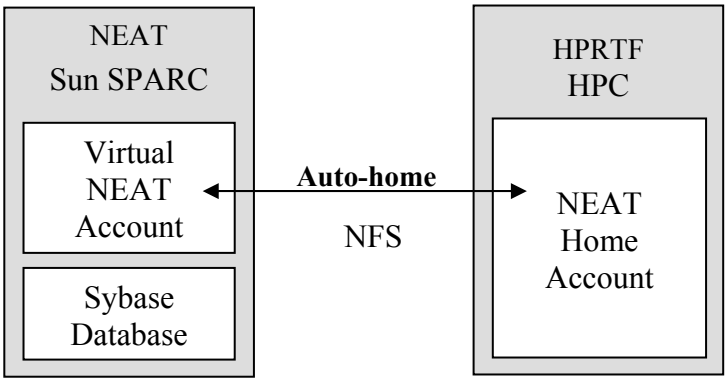


Figure 39 – Physical Storage of NEAT Account and Sybase Database

NEAT Host-Side Install

This step is required if the HPRTF demonstration mode is desired, where the original NEAT tool suite interacts in real-time with the HPRTF HPC process. NEAT requires installation onto a Sun SPARC host, running Sybase 12+ and Solaris 2.6 or greater. If only stand-alone operation is needed, this step can be skipped.

- Create a NEAT user account. If possible, create the account on the HPC side and share it with the Sun host using the Unix "auto-home" feature. Make sure the account has the path "/home/neat" to avoid unnecessary configuration steps.

- Install Sybase. Insert the Sybase CDROM and follow installation procedures as directed by the Sybase install program. Follow operating-system specific procedures defined in the Sybase install manual. When prompted to install a database, configure a server named "SYBASE". Entire the path of the database file as "/usr/sybase/databases/NEAT" and the sys-proc file "/usr/Sybase/databases/SYSPROC". Use default settings for all other fields.

- Insert the HPRTF Software CDROM into the target Sun machine. From the NEAT user base directory (/home/neat), open the "NEAT Software" tar distribution.

  cd /home/neat
  tar xvf /cdrom/cdrom0/NEAT_host_software.tar

  Installation of the source into the base directory of NEAT is necessary to avoid configuration problems in NEAT. If it is not possible to maintain this directory path, the ".neat" configuration script will need to be modified to the new path.

  The NEAT host-side install will create a base directory "HPRTF" with the NEAT host-side install contained within it.

- Test the NEAT file path configurations.

  cd HPRTF/NEAT
  source .neat

This will execute the NEAT setup script and initialize the Sun host-side tool suite. The script defines easily readable directory paths at the start of the file, which are used through out the rest of the setup. If any of these paths are invalid, the script returns errors stating the missing components.

- Test the Sybase setup. First test whether the server is already active. This is accomplished by executing the script (in ~/HPRTF/NEAT):

  showserver

  If no servers are listed, then attempt to start the server using the script:

  startserver

  Typical first-time errors sometimes involve user-permissions to Sybase database files. If errors are encountered, change ownership of the database files to the NEAT user. At anytime when NEAT is not being used, the server can be safely stopped using the command:

  stopserver.

  If necessary, refer to the Sybase install manual to resolve any remaining errors.

- Install a NEAT database schema. Change directory to "~/HPRTF/NEAT/fd" and run the "install" script. This will configure a schema to manage NEAT-specific data.

  cd fd
  install

  Install a NEAT scenario. Change directory to "~/HPRTF/NEAT/scenario" and run the "load_neat_scenario" script. This will load the "neat_scenario" scenario data file into the database. The installation procedure displays the contents of the contact reports in ASCII form as they are read back from Sybase and validated.

  cd ../scenario
  load_neat_scenario

- Run NEAT. After sourcing the ".neat" configuration script, execute the Nodal Aggregation Tool with the command:

  run_nat

  The following windows will appear:

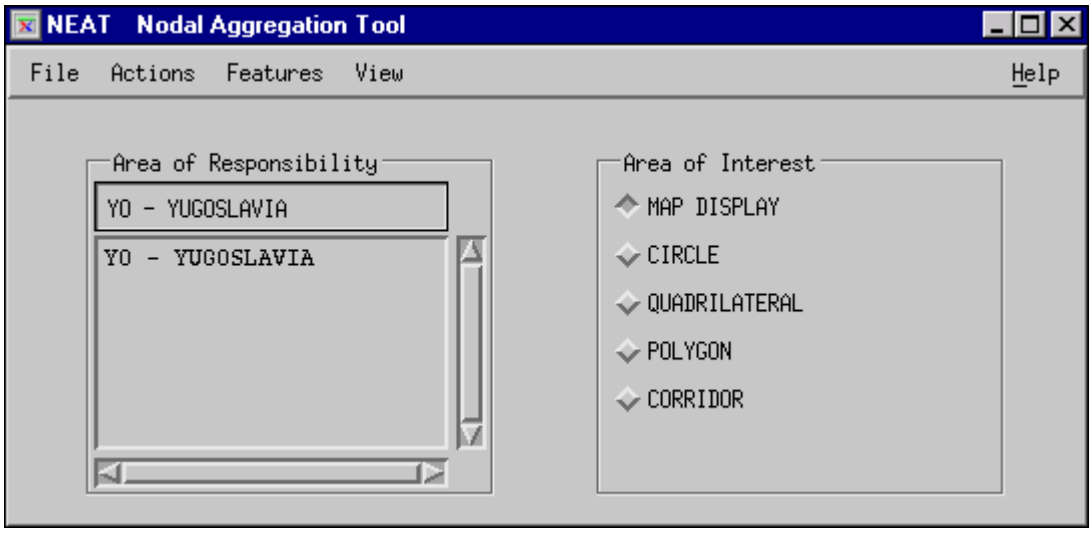


Figure 40 - NAT Main Control Window

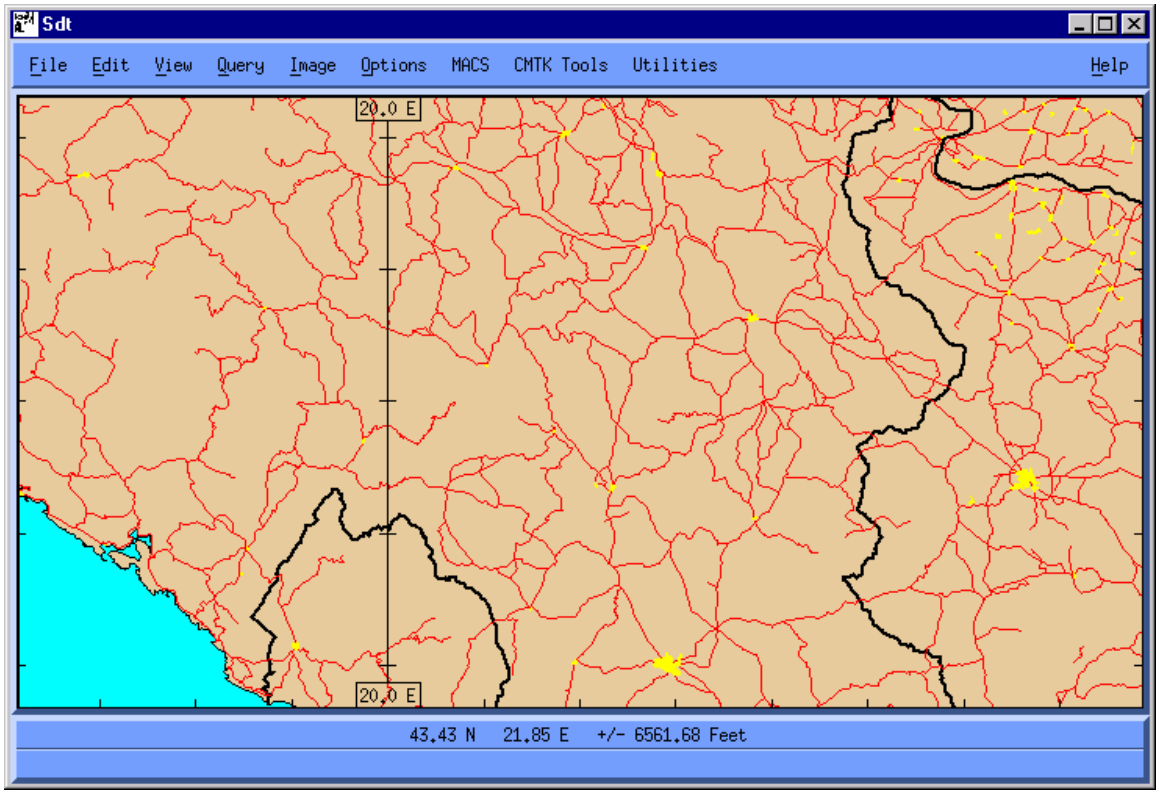


Figure 41 - NEAT Display GUI

At this point the NEAT software is completely installed and has the ability to function either standalone (sequential optimized on the Sun host) or in concert with the HPRTF component. If NAT does not function, observe any error statements from the command line. Errors prohibiting execution include Sybase configuration problems, display settings, and NEAT configurations. All three usually state the problem clearly. All NEAT-related environmental configurations are set from the .neat script and can be edited there if necessary. The map display works on both 8-bit and 24-bit display modes on most all Sun frame buffers.

HPRTF Software Install

Installation of the HPRTF software onto the HPC target host is considerably easier than the NEAT software installation and validation in section B1.1. To guarantee a usable demonstration system the NEAT account on the HPC should be shared with the Sun host containing the NEAT install. If a file-based mode is to be used then this is not an issue. Additionally, an optional connection mode between NEAT and HPRTF allows communication through TCP sockets. This mode should be used when account sharing across machines is not an option.

- Insert the HPRTF Software CDROM into the target Sun machine. From the NEAT user base directory (/home/neat), open the "HPRTF Software" tar distribution.

  cd /home/neat
  tar xvf /cdrom/cdrom0/HPRTF_host_software.tar

  This will install the entire HPRTF software distribution. A directory is created for each Force Aggregator C/C++ library needed for compilation, and a main "hprtf" directory organizes all of the top-level Matlab wrappers, which define the HPRTF system.

This completes the installation process. Compilation directions follow in the next section.

HPRTF Software Compilation

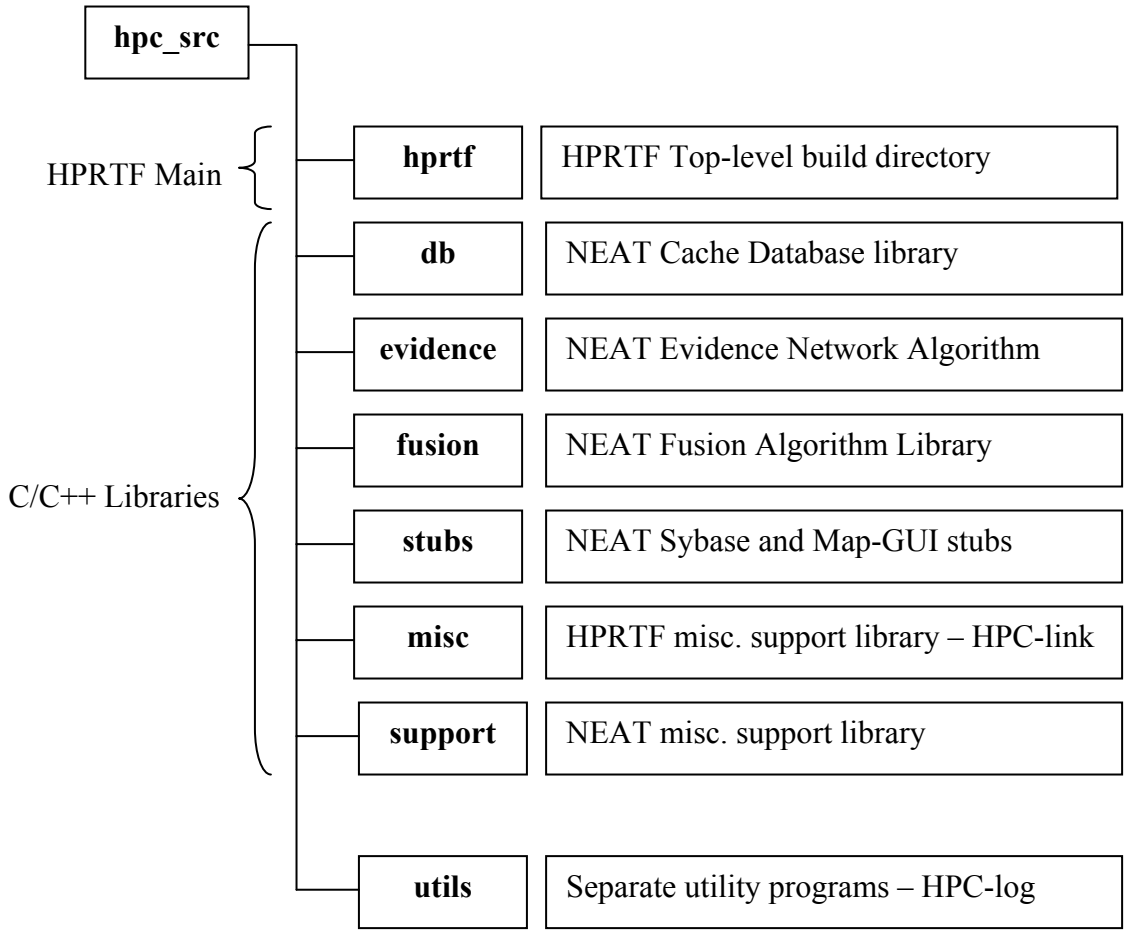


Figure 42 - HPRTF Software Distribution

To compile the HPRTF system, first compile all of the support libraries, build the main source under the "hprtf" subdirectory. All required support libraries can be compiled by using the command "make_neat_libs" under the main "hpc_src" directory.

make_neat_libs

The second step consists of generating and compiling the program main for HPRTF. RTExpress is required for this process.

The main Matlab file containing the top-level Force Aggregation function is the "force_aggregator.m" file in the "hprtf" subdirectory. RTExpress uses this file, as well as machine configurations and user options, to build the HPRTF main source. This is accomplished by using the RTExpress "mapit" utility. As with any other RTExpress utilities, first run the "rtsession" command to establish a license token.

rtsession
mapit force_aggregator

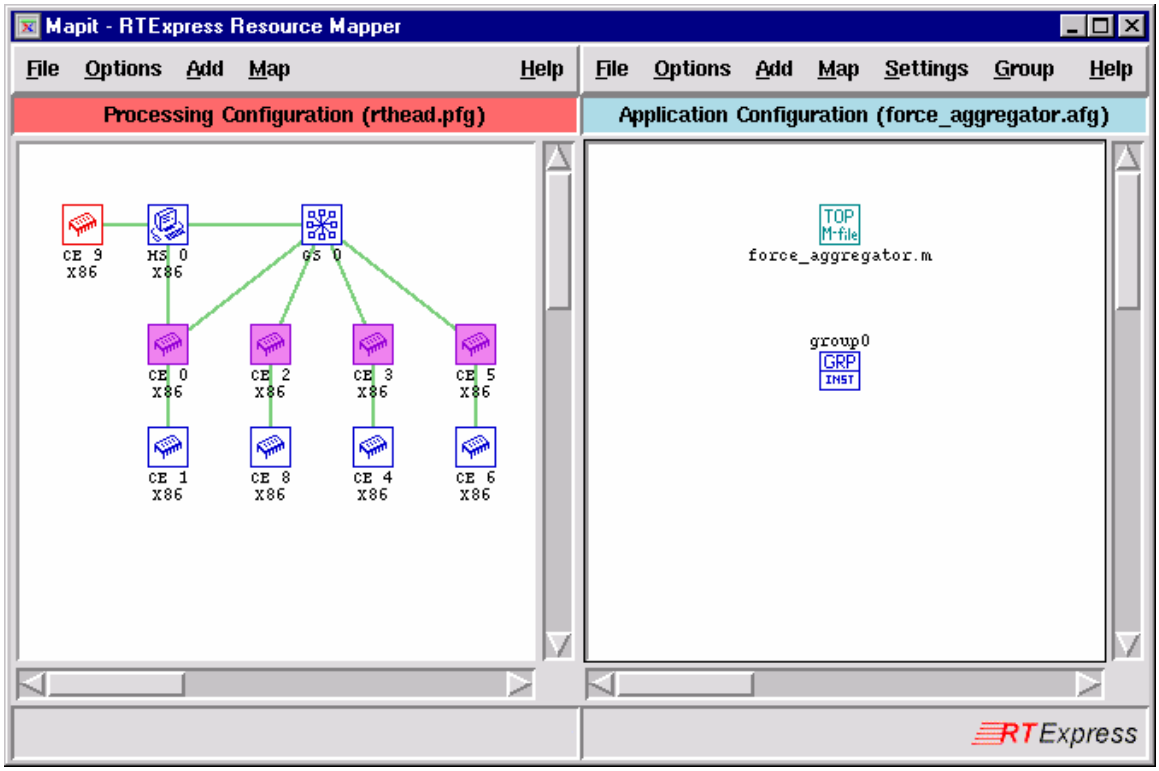


Figure 43 - RTExpress mapit utility

The left-side display shows processor configuration and assignments. The right-side half of the display gives access to compilation options in the HPRTF application. See the RTExpress user manual for more information and details on usage of this tool.

To generate the HPRTF main source, first build the “makefile” and launch-script using the “mapit” utility. Select “Generate Make and Launch” under the “Group” main pull-down

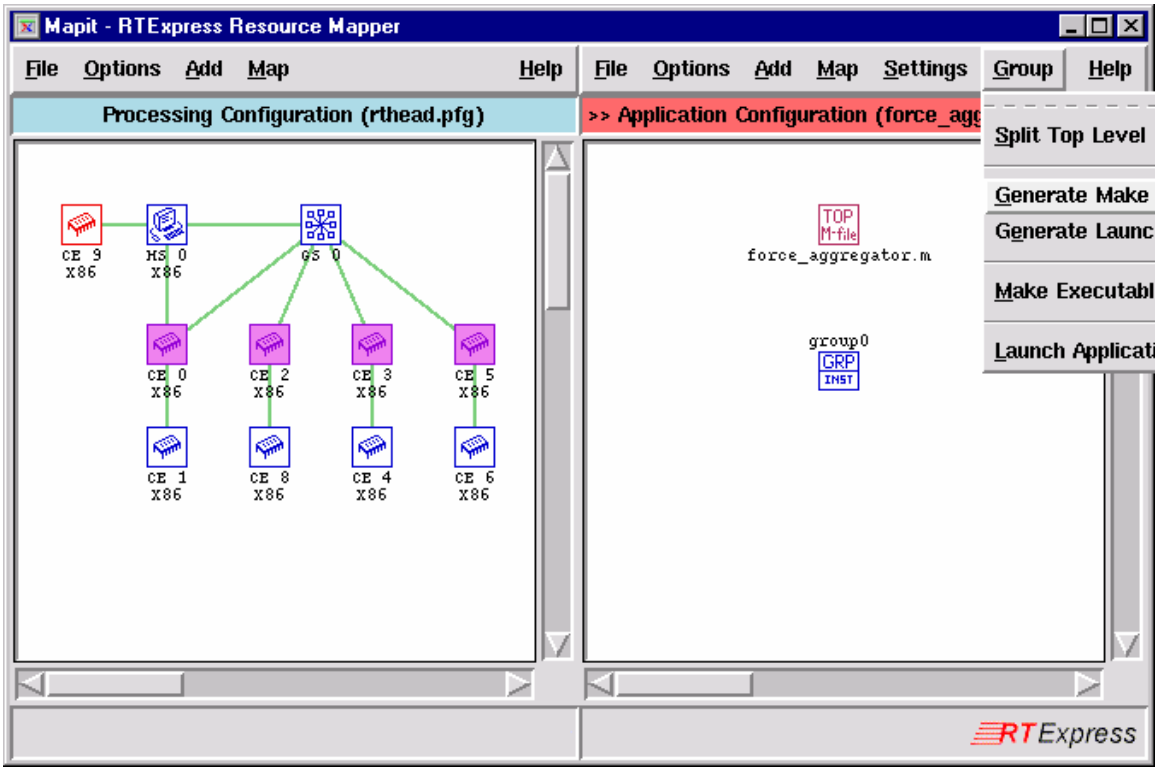


Figure 44 - RTExpress mapit utility – Generate Make and Launch

This generates the “makefile” and a full build environment for compilation of the HPRTF system using the configuration settings specified in the “mapit” GUI. A status window will display the status of each step of this process.

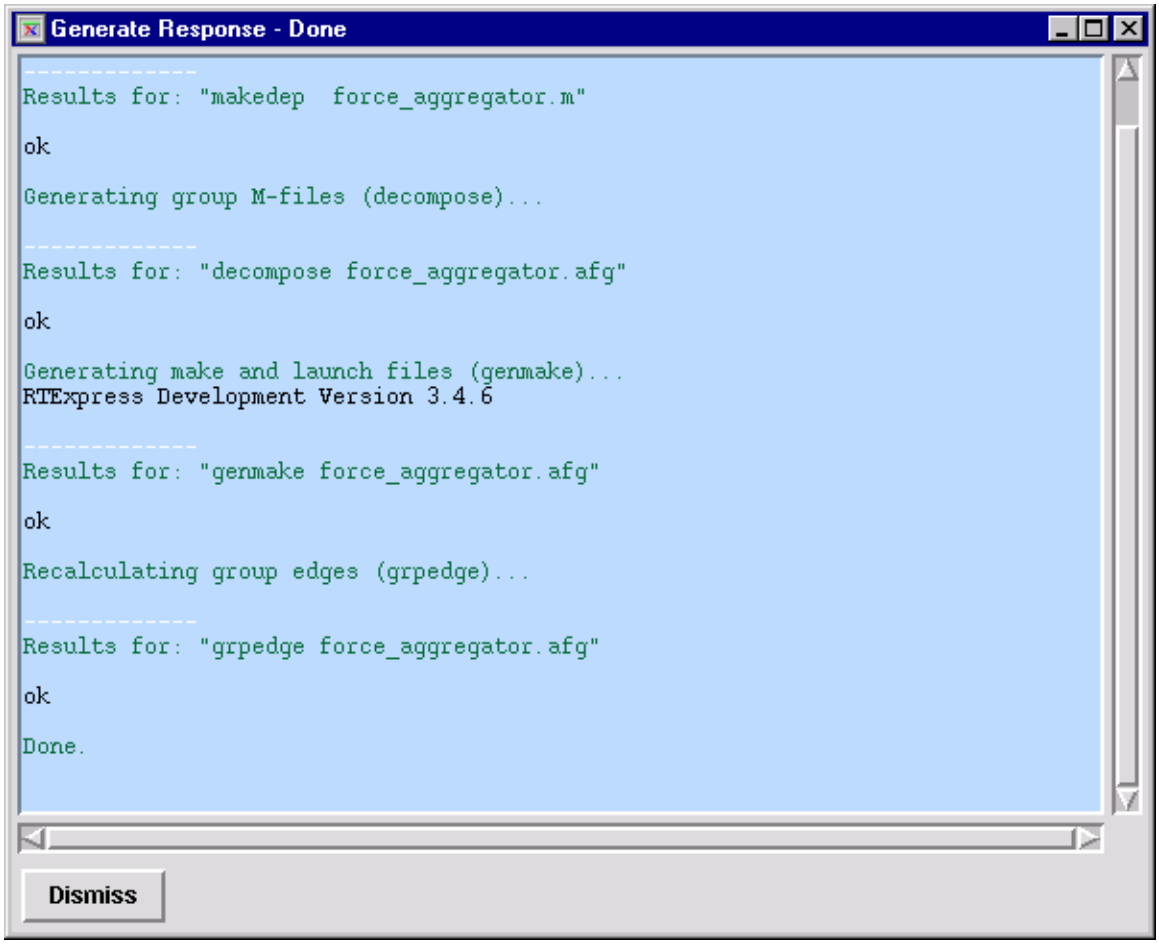


Figure 45 - "Generate Make and Launch" status window

At this point a complete build environment exists. The application can now be compiled from the command-line prompt. A "make" pre-process must be performed on the build environment to allow compilation with the NEAT function libraries.

fixMake
make

This will generate the final executable. Running the "go" script (also generated by "mapit") can now launch the HPRTF application. This will run HPRTF as configured in the "force_aggregator.m" application main.

go

NEAT/HPRTF Demonstration Sequence

This section details what steps are necessary to run NEAT on the Sun host with HPRTF on the HPC host as the Force Aggregator.

- First, prepare to run NEAT on the Sun host.

  cd /home/neat/HPRTF/NEAT
  source .neat

- Make sure the Sybase server is active.

  showserver

- If no servers are listed, start the NEAT Sybase server.

  startserver

- Start the NAT application. There are multiple optional modes for this dual-machine demo. If the NEAT home account has been setup to be shared between machines, the NFS-file mode is preferable. In this case, the HPC hostname and cache filename are necessary command-line arguments. The filename needs to contain the full pathname to assure correct execution.

  run_nat –file <filename> –sig <hpc_host>

  If a shared account is not available, the socket mode can be used. This results in slower execution times as an extra step is required on the HPC side to distribute the data cache.

  run_nat –socket <hpc_host>

  Lastly, if instead it is desired to simply generate a cache file for offline testing of the HPRTF system (later using the file as input on the HPC-side), the file-generation option can be used. This will write the file without any processing.

  run_nat –file <cache_output_filename>

The NAT application window and map GUI should now be open.

- First select the military template file to be used for processing. This is usually paired with the input scenario and should have a similar filename as the truth scenario. Select “open” under the “file” pull-down. Next select the file from the file-selection pop-up window.

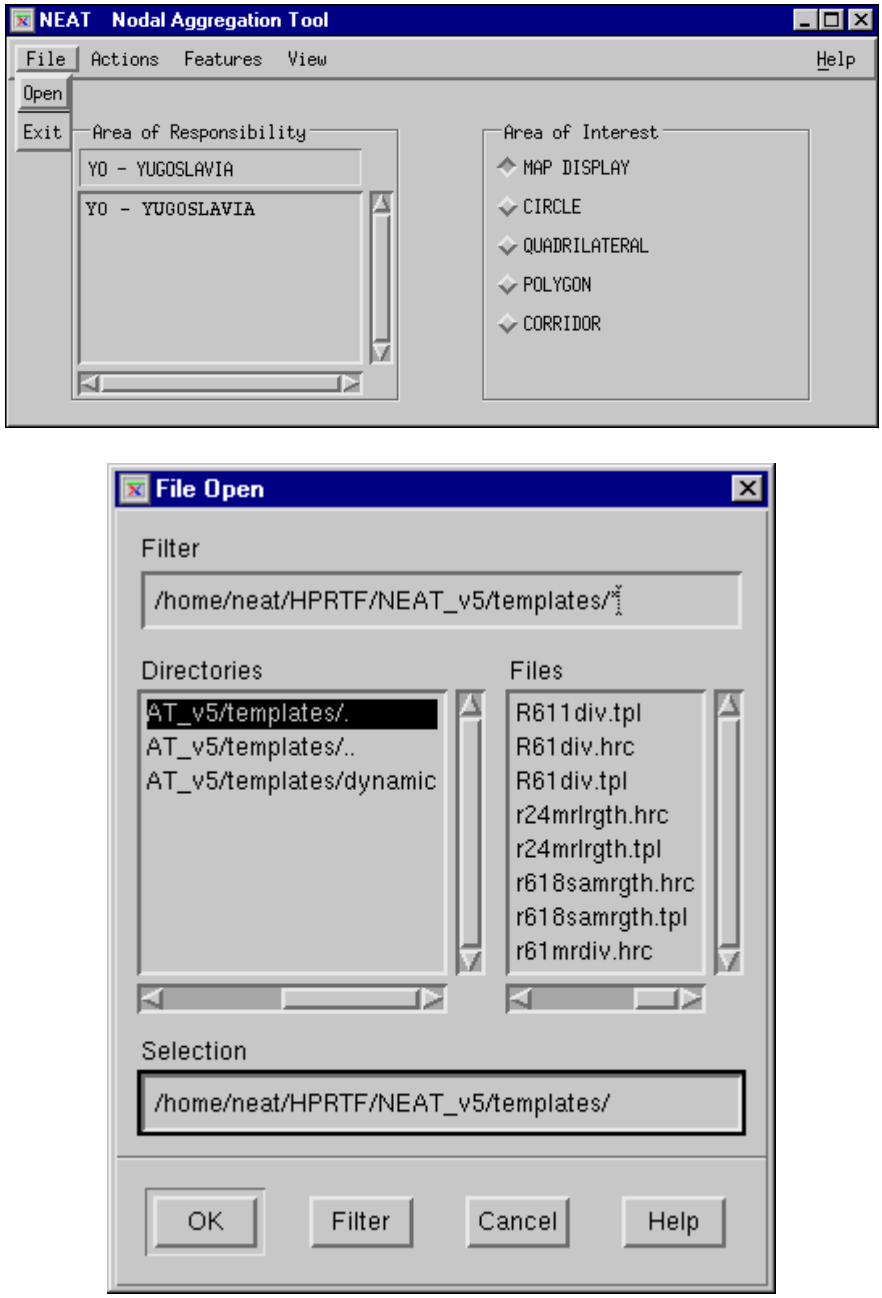


Figure 46 – Loading a Military Template File

- Activate the Force Aggregation function from the NAT window. Select “Military Unit Track” under the “Actions” pull-down.

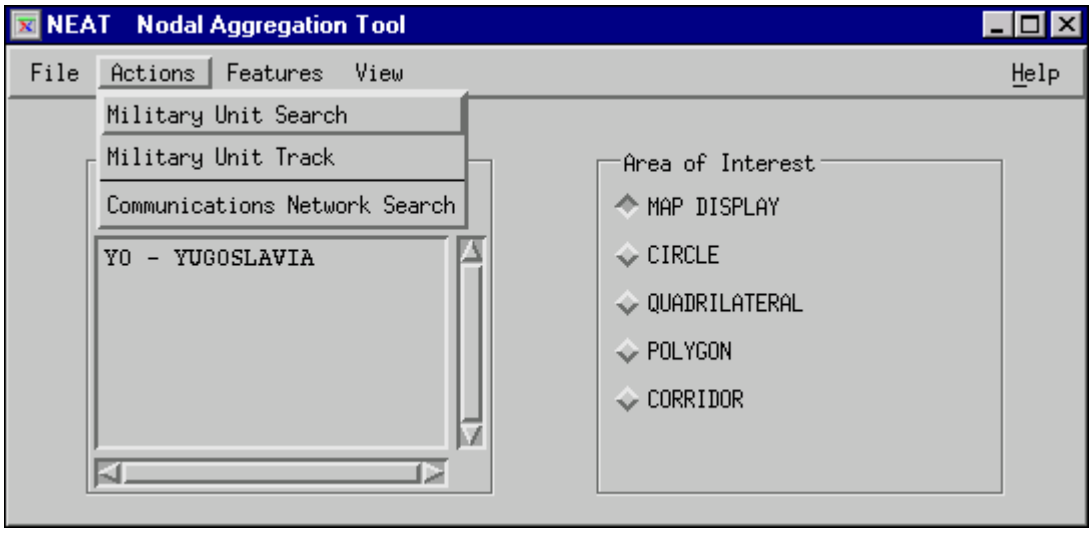


Figure 47 - “Military Unit Track” Action

At this point NEAT will build the contact-report data cache. If the HPRTF system is to be used, and the Force Aggregator has not been started yet on the HPC host, start it now.

- Log onto the HPC and change directory to the main build area “hprtf”. Then run the “go” script to launch the Force Aggregator. The application will then go into a sleep state until the data cache is received from NEAT.

Once NEAT completes building the data cache, the Nodal Track window appears. (Following page)

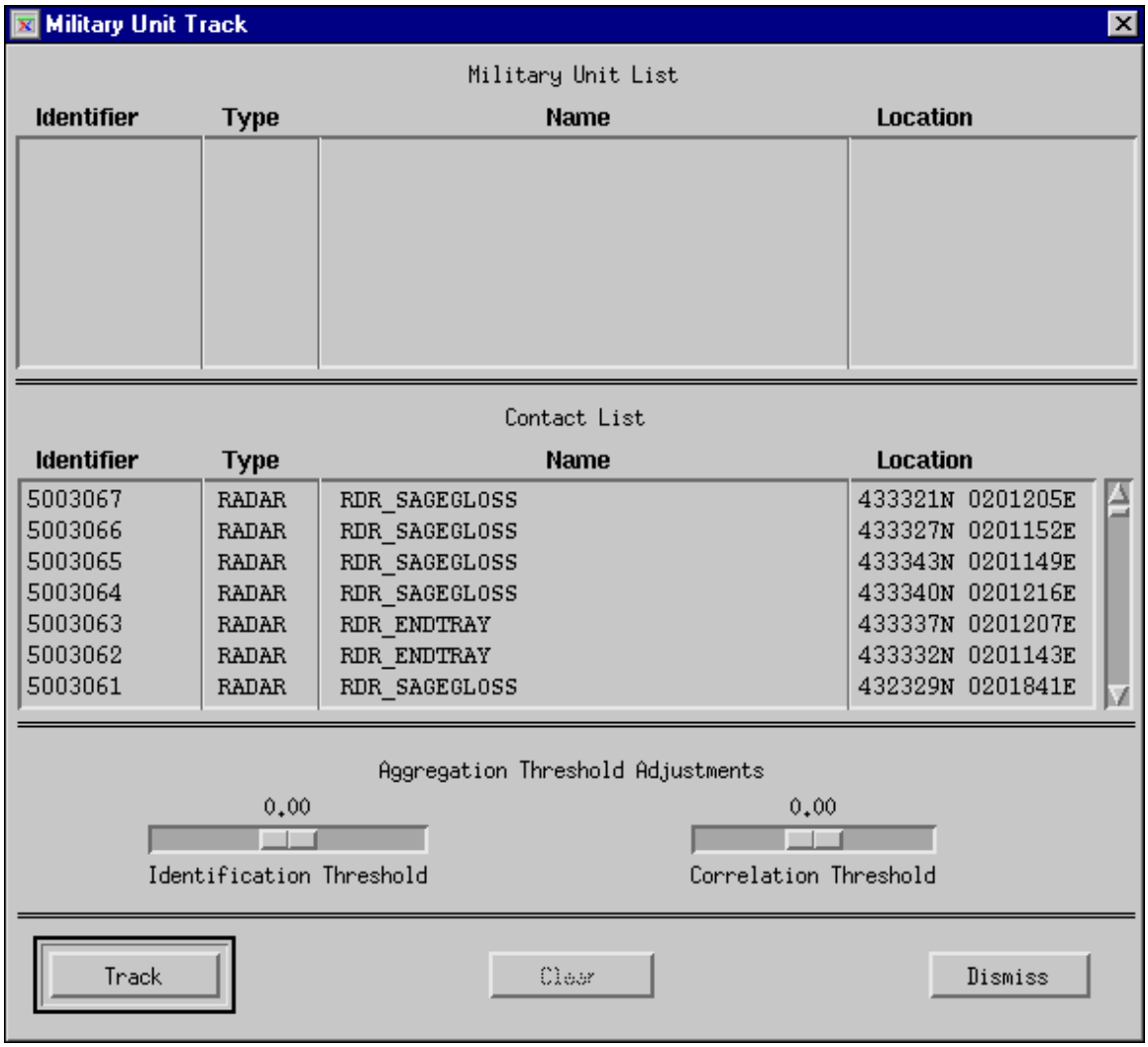


Figure 48 - Nodal Track Window

- Pressing the “track” button at this point sends the data cache to the HPC and starts the Force Aggregation process. The login shell used to start the Force Aggregator should come to life and immediately begin processing. When completed, the results (C2 Nodes) are transmitted back to the NEAT application via NFS file or socket, depending on the mode used.

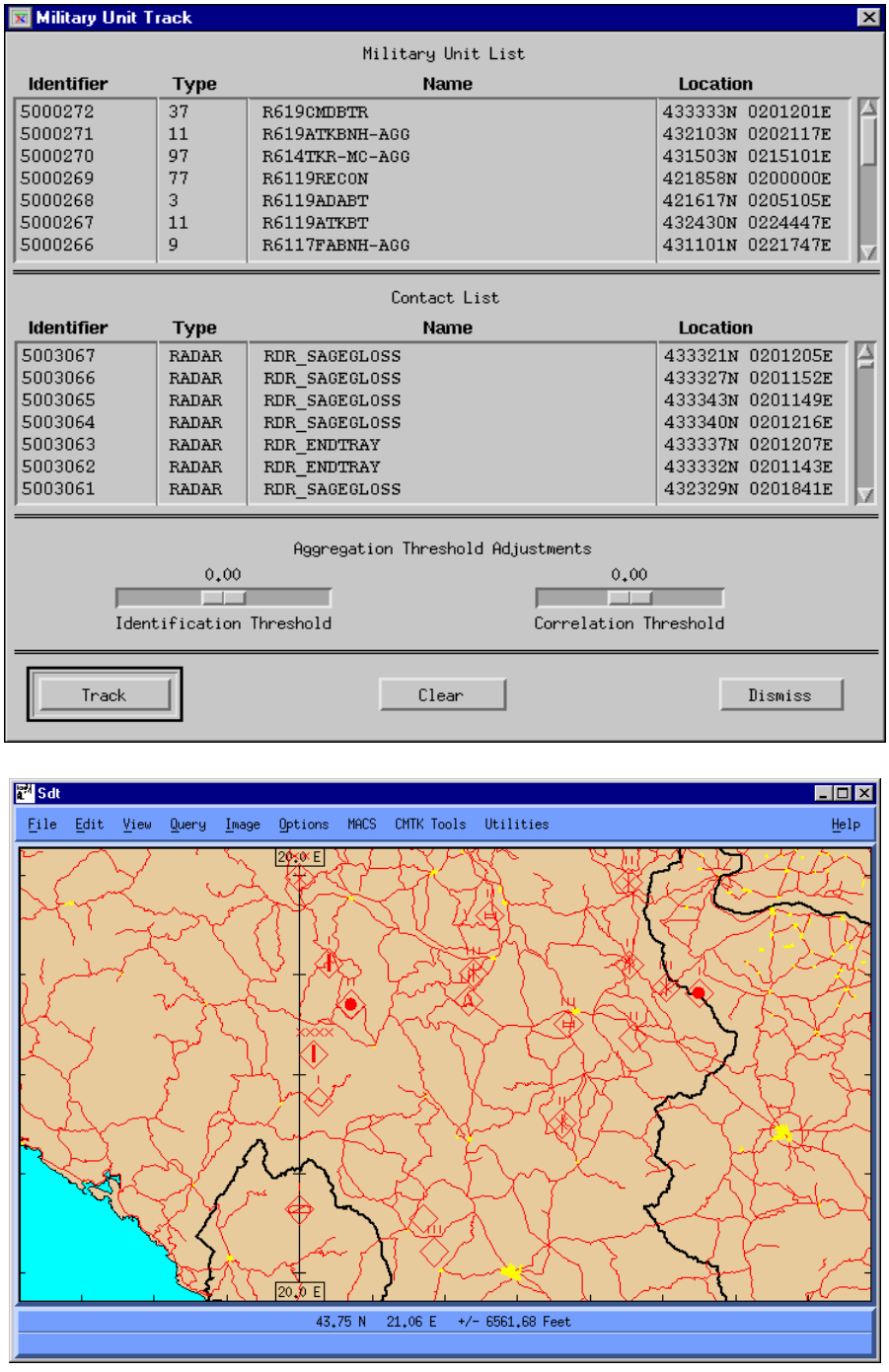


Figure 49 - Force Aggregation Results

This concludes the demonstration sequence. The Force Aggregator can be activated again without restarting the NAT application by selecting the “Track” button again. At anytime after or between processing C2 Nodes, the results from a previous run can be deleted from the screen and Sybase database by selecting the “clear” button.